CHANDU

1) What is JAVA :-

- Java is a programming language and computing platform that was first released by Sun Microsystems in 1995.
- It is used for developing and deploying a wide range of applications, including mobile applications, web applications, and games.

★Features of JAVA :-

- Simple Language.
- Secure.
- Portable.(Compile at one computer and run at any computer.)
- Robust. (A robust program is one that is resistant to errors and can continue to function correctly even when faced with unexpected situations.)
- **Robust** :-

I. Exception Handling :-

- Java has a built-in exception handling mechanism that allows you to handle runtime errors and other exceptional conditions in a controlled way**.**

II. Memory Management :-

- Java uses automatic garbage collection to manage memory, which means that you do not have to worry about manually allocating and freeing memory.

★ Platform-independent language :-

- Java is considered a platform-independent language because it is designed to run on any device that has a Java Virtual Machine (JVM). This means that you can write a Java program on one device (e.g., a Windows computer), and then run it on a different device (e.g., a Mac, Linux,) without making any changes to the code.

★ Byte Code :-

- byte code refers to the instructions that are executed by the Java Virtual Machine (JVM) at runtime.
- When you write a Java program, it is compiled into bytecode, which is a set of instructions that can be run on any platform that has a JVM.

2) Object Oriented Program (OOPS) :

- It is a programming paradigm based on the concept of "objects", which can contain data and code that manipulates that data.
- Main focus is on "Class and Object".
- Reduce the complexity of the program.
- Code Reusability.
- **Features Of OOP is (** The main principles of OOPS in Java are:)

I. Inheritance
II. Polymorphism
III. Abstraction
IV. Encapsulation

I. Inheritance :-

- Creating a new class from already existing class.
- The new class is called the subclass, and the existing class is the superclass.
- The subclass inherits characteristics from the superclass and can also have additional characteristics of its own**.**
- In Java, inheritance is implemented using the **"extends"** keyword, which allows a subclass to inherit all the properties and behavior of its superclass.
- Inheritance provides several benefits, including:
  1. **Reusability**: Inheritance allows us to reuse existing code by creating new classes that inherit properties and behavior from existing classes.
  2. **Abstraction**: Inheritance allows us to create abstract classes that provide a high-level view of the system and can be extended by concrete subclasses.
  3. **Polymorphism**: Inheritance allows us to create subclasses that can override or extend the behavior of the superclass, which enables polymorphic behavior in the code.

★ Types of Inheritance ?

- They are 4 types :
  1. **Single Inheritance.**
  2. **Multilevel Inheritance.**
  3. **Hierarchal Inheritance.**
  4. **Multiple Inheritance.**

1. Single Inheritance :-

- a class to inherit properties and methods from another class.
- Single inheritance is a type of inheritance in which a class derives from a single superclass.
- This means that the subclass has access to all of the public and protected members of the superclass, and can add its own members.
- **Example:**

```
class A
{
            int p;
            A(){}       //empty constructor
            A(int a)   //single parameter constructor
            {
                p=a;
            }
             void display()
             {
                System.out.println("Iam base class method dispaly.my variable p value is :"+p);
             }
}
class B extends A
{
            int q;
            B(){}
            B(int a, int b)
            {
                p = a;
                q = b;
            }

            void show()
            {
                    System.out.println("iam detived class method show.my variable q value is:"+q);
                }
}
class Inheritance{
                public static void main(String[] arr) {

                    B value = new B(25,40);
                    value.show();

                    value.display();
                }
}
```

2. **Multilevel inheritance :-**

- If a subclass derives from a superclass, that subclass derives another subclass is called Multilevel inheritance.

```
class A
        {
                int p;
                A(){}   //empty-Constructor
                A(int p)
                {
                        this.p=p;
                }
                void display(){ }
        }
class B extends A
        {
                int q;
                B(){}
                B(int p,int q)
                {
                        this.p=p;
                        this.q=q;
                }
                void show() { }
        }
class C extends B
        {
                int r;
                C(){}
                C(int p,int q,int r)
                {
                        this.p=p;
                        this.q=q;
                        this.r=r;
                }
                void add()
                {
                        System.out.println("addition of p,q,r is :"+(p+q+r));
                }
        }
class ClassName{

                public static void main(String[] arr) {
                        C obj = new C(55,879,4547);
                        obj.add();
                }
        }
```

3. **Hierarchal Inheritance :-**

- When a superclass has two or more subclasses it is called Hierarchical Inheritance.

```
class A
{
        int p;
        A(){}
        A(int p)
        {
                this.p=p;
        }
        void display() {}
}
}
class B extends A
{
        int q;
        B(){}
        B(int p,int q)
        {
                this.p=p;
                this.q=q;
        }
        void show()
        {
                System.out.println("adding of p,q is:"+(p+q));
        }
}
class C extends A
{
        int r;
        C(){}
        C(int p,int r)
        {
                this.p=p;
                this.r=r;

        }
        void add()
        {
                System.out.println("addition of p,r:"+(p+r));
        }
}
class Hierarchical
{
        public static void main(String[] arr) {
                B obj1 = new B(56,54);
                obj1.show();

                C obj2 = new C(78,45);
                obj2.add();
        }
}
```

4. **Multiple Inheritance** :-

- If a subclass is derived from two or more super classes, it is called multiple inheritance.
- Java does not support multiple inheritance through class extension because it can lead to ambiguities and the confusion of which method or variable to use. Instead, it provides interfaces and delegation as mechanisms for achieving similar functionality.

```
class A
{
        void methodA()
        {
                System.out.println("iam methodA from class A");
        }
}
interface B
{
        void methodB();
}
interface C
{
        void methodC();
}
class D extends A implements B,C
{
        public void methodB()
        {
                System.out.println("iam methodB from class D");
        }
        public void methodC()
        {
                System.out.println("iam methodC from class D");
        }
}
class Multipleinh
{
        public static void main(String[] arr) {

                D ob = new D();
                ob.methodA();
                ob.methodB();
                ob.methodC();
        }
}
```

II. Polymorphism :-

- An operation Exhibits different behaviours in different instances.
- It allows objects of different classes to be treated as objects of the same class.
- It is defined as the ability of an object to take on multiple forms.
- Poly→ many and Morphic→ Behavior. (Polymorphism→ ManyBehavior).{Different-behavior}

★ There are two types of polymorphism in Java:-

1) Compile-time polymorphism

or (static polymorphism)

or Method Overloading :-

- This type of polymorphism is achieved through **method overloading**
- Two or more methods in a class having same name with different parameters.
- At compile-time, the Java compiler determines which method to call based on the number and type of arguments passed in the method call.

```
class MathOperations {
            int add(int a, int b) {
              return a + b;
            }
            double add(double a, double b) {
              return a + b;
            }
            int add(int a, int b, int c) {
              return a + b + c;
            }
          }
class Main {
            public static void main(String[] args) {
              MathOperations math = new MathOperations();
              System.out.println(math.add(2, 3)); //output 5
             System.out.println(math.add(2.5, 3.5));  //output 6.0
              System.out.println(math.add(2, 3, 4)); //output 9
            }
          }
```

- In this example, we have a **MathOperations** class that contains three methods with the same name **add**, but with different numbers of arguments.
- The first **add** method takes two integer arguments and returns their sum. The second **add** method takes two double arguments and returns their sum.
- The third **add** method takes three integer arguments and returns their sum.

2) Run-time polymorphism

or (dynamic polymorphism)

or Method Overriding :-

- This type of polymorphism is achieved through **method overriding.**
- A subclass defines a method with the same name and same parameters as a method in the superclass.
- The subclass method is said to override the superclass method, and it will be used instead of the superclass method when the subclass object is used.
- At runtime, the Java Virtual Machine (JVM) determines which method to call based on the actual type of the object that the method is called on.

Example :-

```
class Animal {
            public void makeSound() {
              System.out.println("The animal makes a sound");
            }
          }

class Dog extends Animal {
            public void makeSound() {
              System.out.println("The dog barks");
            }
          }

public class Main {
            public static void main(String[] args) {
              Animal animal = new Animal();
              animal.makeSound(); // Output: The animal makes a sound

              Dog dog = new Dog();
              dog.makeSound(); // Output: The dog barks
            }
          }
```

- In this example, the Animal class has a method named makeSound that prints "The animal makes a sound". The Dog class extends the Animal class and provides a new implementation for the makeSound method that prints "The dog barks".

- In the main method, we create an instance of the Animal class and call its makeSound method, which prints "The animal makes a sound". Then, we create an instance of the Dog class and assign it to a reference of type Animal. When we call the makeSound method on the dog object, the overridden version in the Dog class is called, which prints "The dog barks".

- This demonstrates how polymorphism allows a single method to take different forms based on the type of the object it is called on.

III. Abstraction :-

- Representing essential features without include background details.
- Abstraction is the process of hiding the implementation details of an object and showing only the necessary information to the user.
- In Java, abstraction is achieved through the use of interfaces and abstract classes. An interface defines a set of methods that a class must implement, while an abstract class can provide a partial implementation of a class.
- Abstraction provides several benefits, including:

  1. **Simplification**: Abstraction helps to simplify complex systems by hiding the implementation details and exposing only the essential features and behavior. This makes the code easier to understand and maintain.

2. **Reusability**: By abstracting the implementation details, it becomes easier to create reusable components that can be used in different parts of the code.
3. **Modularity**: Abstraction allows us to create modular components that can be easily replaced or updated without affecting other parts of the code.

Example:-

```
abstract class Shape {
            abstract void draw();
        }

class Rectangle extends Shape {
            void draw() {
              System.out.println("Drawing a Rectangle");
            }
        }

class Circle extends Shape {
            void draw() {
              System.out.println("Drawing a Circle");
            }
        }

public class Main {
            public static void main(String[] args) {
              Shape rectangle = new Rectangle();
              Shape circle = new Circle();

              rectangle.draw();   //output  Drawing a Rectangle
              circle.draw();      //output  Drawing a Circle
            }
        }
```

- In this example, the **Shape** class is defined as an abstract class with an abstract method **draw**. The **Rectangle** and **Circle** classes extend the **Shape** class and provide concrete implementations for the **draw** method.
- The **Main** class creates objects of the **Rectangle** and **Circle** classes and calls the **draw** method on each of them.
- This example demonstrates how abstraction allows you to create a common interface for different classes and encapsulate their implementation details. The **Main** class does not need to know the specifics of how a rectangle or a circle is drawn. It only needs to know that it can call the **draw** method to draw any shape. This makes the code more flexible and easier to maintain.

IV. Encapsulation :-

- Wrapping/Binding up of Variables(DATA) and methods(CODE) into a single unit. This helps to keep the data safe from outside interference and misuse.
- Is the process of hiding the implementation details of an object from the outside world.
- It is achieved by using access modifiers (such as private, protected, and public) to restrict access to the fields and methods of a class.
- In Java, encapsulation is achieved by declaring variables with private access control and providing public methods to access or modify the data stored in the private variables.

- This way, the implementation details of the class can be hidden from the outside world, and the user of the class can only interact with the class through the public methods.

  Example :-

```
class Employee {
        private String name;
        private int age;

        public String getName() {
          return name;
        }

        public void setName(String name) {
          this.name = name;
        }

        public int getAge() {
          return age;
        }

        public void setAge(int age) {
          this.age = age;
        }
      }

public class Main {
        public static void main(String[] args) {
          Employee employee = new Employee();
          employee.setName("John Doe");
          employee.setAge(30);

          System.out.println("Name: " + employee.getName());
          System.out.println("Age: " + employee.getAge());
        }
      }
```

- In this example, the **Employee** class has two private variables **name** and **age** that represent the name and age of an employee. The class also has **get** and **set** methods that allow you to access and modify these variables.
- The Main class creates an Employee object and sets its name and age using the setName and setAge methods. It then retrieves the values using the getName and getAge methods and prints them to the console
- This example demonstrates how encapsulation allows you to hide the implementation details of a class and provide a controlled way of accessing its data. By making the name and age variables private, you can ensure that they are only modified through the set methods. This makes the code more secure and easier to maintain.

★ Why java is not 100% Object-oriented ?

- Because of primitive datatypes namely : boolean, byte, char, int, float, double, long, short.
- To make them OO we have wrapper class which actually "wrap" the primitive datatype into an object of the class.

3) JVM (Java Virtual Machine) :

- The Java Virtual Machine (JVM) is a virtual machine that enables a computer to run Java programs.
- It provides the environment in which Java programs can run.
  - A. **Class Loader :-** To Load .class files into JVM and scan the program and find errors.
  - B. **Run Time Data Area :-** Load the program and store and create Objects.
  - C. **Heap :-** To elongate the memory to each object.
  - D. **Java.Stack :-** To store temporary data.

★ JDK (Java Development Kit) :

- JDK stands for Java Development Kit. It is a software development environment used for developing Java applications and applets.
- It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

★ JRE (Java Runtime Environment) :

- Java Runtime Environment (JRE) is a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications and applets written in Java programming language.
- JRE is the implementation of JVM which allows the execution of the Java bytecode.
- The JRE is included in the JDK (Java Development Kit) and it is not needed to be installed separately.
- JRE is used to run the Java applications on the computer. It contains libraries, the Java Virtual Machine, and other components necessary to run applets and applications written in the Java programming language.

4) DATA-Types :

- A. **Integer** (int, byte, short, long)
- B. **Float** (float, double)
- C. **Char**
- D. **Boolean**

A. Integer Data Type :-

1) int :-

- "int" is a primitive data type that represents a 32-bit(4-bytes) signed integer.
- It can hold values in the **range (-2^{31} to 2^{31-1})** or **-2147483648 to 2147483647.**
- Default values is 0

2) byte :-

- It is typically used to store small integers or to save memory when working with large arrays of data.
- represents an 8-bit(1-bytes) signed integer.
- It can hold values in the **range -128 to 127.**
- By default, a byte variable is initialized to 0.

3) short :-

- It can be useful for representing small numbers, such as the number of items in a small list or the number of days in a small time period.
- It uses 16-bits(2-bytes) of memory.
- It can hold values in the **range -32768 to 32767.**

4) long :-

- It is used to represent a larger range of whole numbers than the int data type. The long data type can be declared by using the keyword "long".

B. Float Data Type :-

1) float :-

- “float” is a primitive data type that represents a single-precision 32-bit(4-bytes) floating-point number.
- It can hold values ranging from approximately **1.4E-45 to 3.4028235E38**, with a precision of about 7 decimal digits.(After Decimal point 7 numbers)
- The default value of a float variable is 0.0f.

2) double :-

- the double data type is used to represent a double-precision floating-point number, which is a number with a decimal point.
- A precision of about 15 decimal digits. .(After Decimal point 15 numbers)

C. Char Data Type :-

- In Java, a char is a primitive data type that represents a single 16-bit Unicode character.
- It can hold any value from the Unicode character set
- Default value is Null**.**

D. Boolean Data Type :-

- True or False
- Default value is false**.**

★ How many types of data types in java ?

- In Java, there are two main types of data types:

1. **Primitive data types:**
2. **Non-primitive data types:**

1. Primitive datatypes :-

- In Java, there are 8 primitive data types:
- These are the basic data types that are built-in to the Java language.

1. int :-

- This data type is commonly used for storing whole numbers and can be used in mathematical operations such as addition, subtraction, multiplication, and division.

2. **byte :-**
    - It is typically used to store small integers or to save memory when working with large arrays of data.
    - By default, a byte variable is initialized to 0.
3. **short :-**
    - It can be useful for representing small numbers, such as the number of items in a small list or the number of days in a small time period.
    - It uses 2 bytes of memory.
4. **long :-**
    - It is used to represent a larger range of whole numbers than the int data type. The long data type can be declared by using **the keyword "long".**
5. **float :-**
    - The float data type should be used when floating-point values must be used, but the memory footprint needs to be smaller and the precision can be limited.
    - The default value of a float variable is 0.0f.
6. **double :-**
    - the double data type is used to represent a double-precision floating-point number, which is a number with a decimal point.
7. **char :-**
    - used to store a single character value.
    - It uses 2 bytes of memory.
    - It is enclosed in single quotes, for example 'a' or 'B'.
    - It is commonly used to store individual characters in a string.
8. **boolean :-**
    - The data type that can only hold two values: true or false.
    - It is used to represent a logical state, such as true or false, yes or no, on or off, etc.

2. Non-primitive data types :-

- These are also known as reference types and include arrays, classes, interfaces, and strings.
- These types are not built-in to the Java language and are created by the programmer.

5) Operators :

- In Java, an operator is a symbol that performs an operation on one or more operands.
- Operators are used to perform operations such as assignment, arithmetic, comparison, and logical operations.
- Java has the following types of operators:

1) Arithmetic
2) Assignment
3) Relational
4) Logical
5) Conditional (Ternary)
6) Unary
7) Bitwise

1) Arithmetic operators :-

- These operators perform basic arithmetic operations, such as addition, subtraction, multiplication, and division.
- For example: **+**, **-**, *****, **/**.

2) Assignment :-

- These operators assign a value to a variable.
- The most basic assignment operator is the = operator, which assigns a value to a variable.
- For example: **=**, **+=**, **-=**, ***=**, **/=**, **%=**.

3) Relational :-

- These operators compare two values and return a Boolean result.
- For example: **==**, **!=**, **>**, **<**, **>=**, **<=**.

4) Logical :-

- These operators perform logical operations, such as AND, OR, and NOT.
- For example: **&&**, **||**, **!**.
- **&& (AND) -** The && operator returns true if both operands are true, and false otherwise.
- **|| (OR) -** The || operator returns true if either operand is true, and false otherwise.
- **! (NOT) -** The ! operator returns true if its operand is false, and false if its operand is true.

5) Conditional (Ternary) :-

- the conditional operator (also known as the ternary operator) is an operator that provides a shorthand way of writing an if-else statement.
    - The syntax is :

condition ? expression1 : expression2

where "**condition**" is a boolean expression, and "**expression1**" and "**expression2**" are expressions of any type. If the condition is true, the operator returns expression1; otherwise, it returns expression2.

- These operators are used to test a condition and return a value based on the result of the test.
- The most basic conditional operator is the ternary operator (**? :**).

6) Unary :-

- A unary operator is an operator that operates on a single operand.
- In Java, unary operators include:

1. **Unary plus (+) :** indicates a positive value of the operand.

```
int z = -5;
System.out.println(+z); // Output: -5
```

2. **Unary minus (-) :** This operator negates the value of the operand.

```
int a = 10;
System.out.println(-a); // Output: -10
```

3. **Logical negation (!) :** inverts the boolean value of the operand.

```
boolean b = true;
```

```
System.out.println(!b); // Output: false
```

4. **Bitwise complement (~) :** This operator inverts the bits of the operand.

```
int c = 5;
System.out.println(~c); // Output: -6
```

5. **Prefix increment (++) :**

- As you can see, the prefix increment operator (++) is placed before the operand x, and it increments the value of x by 1 before the expression is evaluated. The value of x after the increment is 6.

```
public class Main {
        public static void main(String[] args) {
          int x = 5;
          System.out.println("Before increment: " + x);    //output Before increment: 5
          System.out.println("Result of ++x: " + ++x);     //output is  Result of ++x: 6
          System.out.println("After increment: " + x);     //output is  After increment: 6
        }
      }
```

6. **Prefix decrement (--):**

- As you can see, the prefix decrement operator (--) is placed before the operand x, and it decrements the value of x by 1 before the expression is evaluated. The value of x after the decrement is 4.

```
public class Main {
        public static void main(String[] args) {
          int x = 5;
          System.out.println("Before decrement: " + x);  //Output is : Before decrement: 5
          System.out.println("Result of --x: " + --x);    //Output is : Result of --x: 4
          System.out.println("After decrement: " + x);   //Output is : After decrement: 4
        }
      }
```

7. **Postfix increment (++):**

- As you can see, the postfix increment operator (++) is placed after the operand x, and it increments the value of x by 1 after the expression is evaluated. The value of x after the increment is 6.

```
public class Main {
        public static void main(String[] args) {
          int x = 5;
          System.out.println("Before increment: " + x);  //Output : Before increment: 5
          System.out.println("Result of x++: " + x++);   //Output : Result of x++: 5
          System.out.println("After increment: " + x);   //Output : After increment: 6
        }
      }
```

8. **Postfix decrement (--):**

- As you can see, the postfix decrement operator (--) is placed after the operand x, and it decrements the value of x by 1 after the expression is evaluated. The value of x after the decrement is 4.

```
public class Main {
            public static void main(String[] args) {
              int x = 5;
              System.out.println("Before decrement: " + x);   //Output is : Before decrement: 5
              System.out.println("Result of x--: " + x--);       //Output is : Result of --x: 5
             System.out.println("After decrement: " + x);     //Output is : After decrement: 4
            }
          }
```

7) Bitwise :-

- A bitwise operator in Java performs an operation on the binary representation of integers. or to convert to binary language.
- The following are the bitwise operators in Java

  - **Bitwise AND (&)**
  - **Bitwise OR (|)**
  - **Bitwise XOR (^)**
  - **Left Shift (<<)**
  - **Right Shift (>>)**
  - **Unsigned Right Shift (>>>)**

- These operators are used to manipulate individual bits in an integer value and are commonly used in low-level programming and optimization.

6) Control Statements :-

- control statements are used to control the flow of execution of a program.
- There are three types of control statements:

  A. Conditional statements.
  B. Loop statements. (Iteration Statements)
  C. Branching statements.

A. Conditional Statements :-

- These statements allow a program to execute a certain block of code only if a certain condition is met.
- The two main types of conditional statements in Java are

1. if statement.

- Evaluates a boolean expression and executes a block of code if the expression is true.

  Example :-

```
import java.util.Scanner;
public class IfStatementExample {
         public static void main(String[] args) {
          Scanner input = new Scanner(System.in);

          System.out.print("Enter a number: ");
          int number = input.nextInt();

          if (number > 0) {
            System.out.println(number + " is positive.");
          }
         }
}
```

- In this example, the program prompts the user to enter a number. The number is then stored in the **number** variable. The **if** statement checks if the value of **number** is greater than 0. If it is, the program prints a message saying that the number is positive.

2. if-else statement

```
import java.util.Scanner;

public class IfElseStatementExample {
         public static void main(String[] args) {
          Scanner input = new Scanner(System.in);

          System.out.print("Enter a number: ");
          int number = input.nextInt();

          if (number > 0) {
            System.out.println(number + " is positive.");
          } else {
            System.out.println(number + " is not positive.");
          }
         }
        }
```

- In this example, the program prompts the user to enter a number. The number is then stored in the number variable. The if statement checks if the value of number is greater than 0. If it is, the program prints a message saying that the number is positive. If not, the program prints a message saying that the number is not positive.

3. switch statement.

- Matches a value against multiple cases and executes the corresponding block of code.

```
import java.util.Scanner;

public class SwitchStatementExample {
        public static void main(String[] args) {
         Scanner input = new Scanner(System.in);

         System.out.print("Enter a number: ");
         int number = input.nextInt();
         switch (number) {
          case 1:
           System.out.println("One");
           break;
          case 2:
           System.out.println("Two");
           break;
          case 3:
           System.out.println("Three");
           break;
          default:
           System.out.println("Other");
         }
        }
       }
```

- In this example, the program prompts the user to enter a number. The number is then stored in the **number** variable. The **switch** statement checks the value of **number** against various **case** values (1, 2, 3, etc.). If a match is found, the corresponding message is printed. If no match is found, the program executes the code in the **default** block and prints a message saying "Other".

B. Loop Statements :-

- These statements allow a program to execute a block of code repeatedly, either a fixed number of times or until a certain condition is met.
- The main types of loop statements in Java are

1. for loop

- A for loop in Java is a control structure that allows code to be executed repeatedly for a specific number of times or based on a given condition.
- The syntax for **for-loop** is:

```
for (initialization; condition; increment/decrement (iteration))
{
 // code to be executed
 }
```

The for loop consists of three parts:

- **Initialization:** The starting value of the loop counter is set.
- **Condition:** The loop continues to execute as long as the condition is true.
- **Increment/Decrement:** The loop counter is incremented or decremented after each iteration of the loop.

```
public class ForLoopExample {
        public static void main(String[] args) {
          for (int i = 1; i <= 10; i++) {
            System.out.println(i);
          }
        }
     }
```

- In this example, the **for** loop is used to print the numbers from 1 to 10. The loop uses a counter variable **i** that starts at 1 and is incremented by 1 on each iteration of the loop. The condition **i <= 10** is checked before each iteration of the loop, and the loop continues to execute as long as this condition is true.
- The for loop runs for as long as the condition is true, executing the code inside the loop for each iteration. When the condition becomes false, the loop terminates and control is transferred to the next statement after the loop.

↵ What is enhanced for loop in java ?

- The "enhanced for loop" in Java is also known as a "for-each loop."
- It is a type of loop that is used to iterate over elements of an array or a collection.
- It provides a simple and concise way to access each element of an array or collection without using an index.

```
for (data_tye variable : array_or_collection){
    //code to be executed for each element
}
```

Example :-

```
public class EnhancedForLoopExample {
                public static void main(String[] args) {
                  int[] numbers = {1, 2, 3, 4, 5};

                  for (int number : numbers) {
                    System.out.println(number);
                  }
                }
}
```

- In this example, the enhanced **for** loop is used to iterate over an array of integers called **numbers**. The loop uses the **number** variable to represent each element of the **numbers** array in turn, and the code inside the loop body is executed for each iteration of the loop. In this case, the code inside the loop body simply prints the value of **number**. The result of this code will be the numbers 1 to 5 being printed, one per line.

2. while loop

- A while loop in Java is a control flow statement that repeatedly executes a block of code as long as the specified condition is true.
- ➢ The syntax for while loop is:

```
while (condition)
{
    //code to be executed
}
```

- The condition is evaluated before each iteration of the loop. If the condition is true, the code inside the loop is executed. If the condition is false, the loop terminates and control is transferred to the next statement after the loop.

```
import java.util.Scanner;

public class WhileLoopExample {
        public static void main(String[] args) {
         Scanner input = new Scanner(System.in);

         int number = 0;

         System.out.print("Enter a positive number: ");
         number = input.nextInt();

         while (number <= 0) {
           System.out.println("Invalid input. Please enter a positive number.");
           System.out.print("Enter a positive number: ");
           number = input.nextInt();
         }
         System.out.println("You entered a positive number: " + number);
        }
}
```

- In this example, the program prompts the user to enter a number. The number is stored in the **number** variable. The **while** loop checks the value of **number** to see if it is less than or equal to 0. If it is, the program prints an error message and prompts the user to enter a new number.
- This continues until the user enters a positive number, at which point the loop terminates and the program prints a message saying that a positive number was entered.

What is nested while loop in java ?

- A nested while loop in Java is a loop inside another loop.
- The inner loop is executed completely for each iteration of the outer loop.
- The inner loop runs to completion each time the outer loop runs, creating a pattern of repeated execution.

```
public class NestedWhileLoopExample {
        public static void main(String[] args) {
         int i = 1;
         int j = 1;

         while (i <= 5) {
           j = 1;
           while (j <= i) {
             System.out.print("*");
             j++;
           }
           System.out.println();
           i++;
         }
        }
       }
```

- In this example, two **while** loops are used to print a pattern of asterisks to the console. The outer loop uses the counter variable **i** to control the number of rows in the pattern, and the inner loop uses the counter variable **j** to control the number of asterisks in each row. On each iteration of the outer loop, the inner loop is executed to print a line of asterisks, and the outer loop continues until **i** is greater than 5. The result of this code will be a pattern of asterisks that form a right-angled triangle.

4. do-while loop

- A do-while loop is similar to a while loop, but with a key difference: the loop's code block is executed at least once before the condition is checked.
- The syntax for a do-while loop is:

```
do
 {
    // code to be executed
 }
while (condition);
```

- The code inside the loop is executed first, and then the condition is evaluated. If the condition is true, the loop continues to execute. If the condition is false, the loop terminates and control is transferred to the next statement after the loop.

```
import java.util.Scanner;

public class DoWhileLoopExample {
        public static void main(String[] args) {
          Scanner input = new Scanner(System.in);

          int number = 0;

          do {
            System.out.print("Enter a positive number: ");
            number = input.nextInt();

            if (number <= 0) {
              System.out.println("Invalid input. Please enter a positive number.");
            }
          } while (number <= 0);

          System.out.println("You entered a positive number: " + number);
        }
      }
```

C. Jump Statement OR Branching Statements :-

- These statements allow a program to jump to a different point in the code and execute from there.
- The main types of branching statements in Java are :

break statement :-

- The **break** statement is a control flow statement in Java that allows you to immediately exit a loop or a switch statement.
- It is often used to break out of a loop early when a certain condition is met.

```
public class BreakExample {
  public static void main(String[] args) {
    for (int i = 1; i <= 10; i++) {
      if (i == 5) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

- In this example, the **for** loop iterates from 1 to 10 and prints each value of **i** to the console. However, when **i** becomes 5, the **break** statement is executed, causing the loop to exit immediately and the rest of the loop body is not executed. As a result, the numbers 1 to 4 are printed to the console, and the number 5 is not printed.

NOTE :- It is important to note that the **break** statement can only be used inside a loop or a switch statement, and cannot be used outside of these constructs. If you try to use a **break** statement outside of a loop or a switch statement, you will get a compiler error.

➢ continue statement :-

- The **continue** statement is a control flow statement in Java that allows you to skip the current iteration of a loop and continue with the next iteration.
- The **continue** statement is often used in loops to skip over certain iterations based on some condition.

```
public class ContinueStatementExample {
  public static void main(String[] args) {
    for (int i = 1; i <= 10; i++) {
      if (i % 2 == 0) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

In this example, the continue statement is used inside a for loop to skip over even numbers. The loop prints the values of all odd numbers from 1 to 10. On each iteration of the loop, if the value of i is even, the continue statement is executed, which causes the loop to skip to the next iteration without executing the code to print the value of i.

NOTE :- It is important to note that the continue statement can only be used inside a loop, and cannot be used outside of a loop. If you try to use a continue statement outside of a loop, you will get a compiler error.

➢ return statement :-

- The **return** statement is a control flow statement in Java that allows you to return a value from a method.
- When a **return** statement is executed, the control flow of the program immediately transfers from the method that contains the **return** statement back to the calling method.

```
public class ReturnStatementExample {
          public static int factorial(int number) {
            if (number == 0) {
              return 1;
            }
            return number * factorial(number - 1);
          }
          public static void main(String[] args) {
            System.out.println(factorial(5));
          }
        }
```

In this example, the return statement is used inside a recursive function to calculate the factorial of a number. The function takes a single argument number, and returns the factorial of number. The return statement is used to terminate the function and return the result to the calling code. In the main method, the factorial function is called with an argument of 5, and the result of 120 is printed to the console.

NOTE :- It is important to note that the **return** statement can only be used inside a method, and cannot be used outside of a method. If you try to use a **return** statement outside of a method, you will get a compiler error. Additionally, you can use the **return** statement without a value if the method is declared as **void**. In this case, the **return** statement is used to return control to the calling method without returning a value.

7) Arrays :-

- Arrays are used to Store group of **OR** Collection of Similar datatype values.
- **Types of arrays are**

A. Single Dimensional OR One-Dimensional array(1-D) :-

- ➢ A one-dimensional is a type of array that has a single row or column of elements.
- ➢ It is the simplest form of an array and is used to store a list of values.

B. Multi-Dimensional array OR (2-D,3-D) :-

- ➢ **2-D Array :-** Two-Dimensional array will store Collection of One-Dimensional array
- ➢ **3-D Array :-** Three-Dimensional array will Store Collection of 2-D arrays.

★ What is index in java ?

- index is a numerical value used to reference an element within an array, a list, or another data structure that stores multiple values.
- The first element in the data structure has an index value of 0, the second element has an index value of 1, and so on. The maximum index value is one less than the number of elements in the data structure.
- For example, in an array of size 10, the index values range from 0 to 9.
- You can use these index values to access, modify, or remove elements in the data structure.

8) String :-

- In Java, a String is a class that represents a sequence of characters and String is also datatype.
- Collection of characters.
- It is not a primitive data type like int, float, or char
- It can be represented by using " "
- By default, Strings are Immutable.
- Default value is Null.
- String is a class and datatype.
- **String Class methods are:-**

1. length():

- ➢ Returns the length of the string.

2. charAt(int index):

- ➢ Returns the character at the specified index.

3. indexOf(int ch):

- ➢ Returns the index of the first occurrence of the specified character.

4. indexOf(int ch, int fromIndex):

- ➢ Returns the index of the first occurrence of the specified character, starting from the specified index.

5. **substring(int beginIndex):**
   - Returns a new string that is a substring of this string. The substring begins at the specified index and extends to the end of the original string.
6. **substring(int beginIndex, int endIndex):**
   - Returns a new string that is a substring of this string. The substring begins at the specified **beginIndex** and extends to the character at index **endIndex - 1**.
7. **toLowerCase():**
   - Returns a new string with all the characters in lower case.
8. **toUpperCase():**
   - Returns a new string with all the characters in upper case.
9. **trim():**
   - Returns a new string with leading and trailing white space removed.

Here is an example of how to use some of these methods:

```
String str = "Hello World!";

int len = str.length();                // len is 12
char ch = str.charAt(0);               // ch is 'H'
int idx = str.indexOf('o');             // idx is 4
String sub = str.substring(6);          // sub is "World!"
String sub2 = str.substring(6, 11);      // sub2 is "World"
String lower = str.toLowerCase();       // lower is "hello
world!"
String upper = str.toUpperCase();      // upper is "HELLO
WORLD!"
String trim = str.trim();               // trim is "Hello World!"
```

total 19 But these 9 are usually used

★ Explain string class in java ?

- The String class in Java is a built-in class that represents a sequence of characters. It is one of the most widely used classes in Java and is used to represent text in a program.
- Some of the key features of the String class are:
- **Immutable:**
  - Once a string object is created, its value cannot be modified. Any operation that appears to modify a string, such as concatenation or replacement, actually creates a new string object with the modified value.
- **Index-based:**
  - Strings are indexed, meaning that each character in a string has a corresponding position, starting from 0.
- **Efficient:**
  - Strings are implemented as a special type of array, known as a string constant pool. This makes string operations, such as concatenation and substring, relatively efficient.
- **String Pool:**
  - Java has a special memory area called the string constant pool. When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object.

- **Methods:**
    - ➢ The String class provides a number of methods for manipulating strings, such as the length() method to determine the number of characters in a string, the charAt() method to access a specific character, the substring() method to extract a portion of a string, the indexOf() method to find the position of a character or substring, and the replace() method to replace a character or substring.

★ Why Strings are immutable?

- Strings are immutable, which means that once a string object is created, its value cannot be modified.
- There are several reasons why strings are designed to be immutable in Java:

i. Safety and security:

- Since strings are often used to hold sensitive information such as passwords and personal data, making them immutable ensures that they cannot be accidentally or maliciously modified.
- This can help prevent security vulnerabilities and data breaches.

ii. Efficient memory usage :-

- When two or more identical strings are created, they share the same memory location.
- This is known as interning. If strings were not immutable, it would require allocating new memory space for every modification.
- This would lead to a higher memory usage and increase in garbage collection.

iii. Hashcode caching :-

- Many Java classes, such as HashMap and HashSet, use the hashcode of an object to determine its position in an internal data structure.
- When a string object's value is changed, its hashcode would also change, which would require the object to be rehashed, this would be an overhead operation.
- To avoid this cost, strings are made immutable, so their hashcode can be cached and reused.

iv. Simplified debugging :-

- Immutable strings make debugging easier, as they don't change their value over time.
- This can help prevent errors caused by unexpected string modifications.

v. Concurrency :-

- Strings are widely used in multithreaded environments.
- If a string's value can be modified by one thread, it can lead to race condition, where multiple threads are trying to update the same resource.
- This can lead to unpredictable results and may cause a bug.
- When a string is immutable, it can be shared safely among multiple threads without the need for locks or other synchronization mechanisms.

★ Immutable vs mutable strings in java ?

1. **String is an immutable data type :-**
   - meaning its value cannot be changed once it has been created. When you want to modify a string, you actually create a new string with the desired value, rather than modifying the original string.
2. **A mutable string :-**
   - can be modified after it has been created. Java provides the StringBuilder and StringBuffer classes to represent mutable strings. These classes provide methods to append, insert, delete or replace characters in a string, allowing you to modify the string in place.

★ StringBuffer :

- The **StringBuffer** class in Java is used to represent a mutable sequence of characters.
- It is supporting synchronization.
- Here are some of the most commonly used **StringBuffer** methods:

  1. **append(String str) :-**
     - Appends the specified string to the end of this string buffer.
  2. **insert(int offset, String str) :-**
     - Inserts the specified string at the specified position in this string buffer.
  3. **delete(int start, int end) :-**
     - Deletes the characters in the specified range from this string buffer.
  4. **replace(int start, int end, String str) :-**
     - Replaces the characters in the specified range with the specified string.
  5. **reverse() :-**
     - Reverses the order of the characters in this string buffer.

Here is an example of how to use some of these methods:

```
StringBuffer sb = new StringBuffer("Hello");

sb.append(" World!");       // sb is now "Hello World!"
sb.insert(5, ", ");          // sb is now "Hello, World!"
sb.delete(5, 7);             // sb is now "Hello World!"
sb.replace(6, 11, "there"); // sb is now "Hello there!"
sb.reverse();                // sb is now "!ereht olleH"
```

★ String builder

- The StringBuilder class in Java is a mutable sequence of characters. Unlike the String class, which is an immutable sequence of characters, the StringBuilder class provides methods to modify a string in place.
- Synchronization no guarantee.
- Here are some common operations you can perform with the StringBuilder class:

  1. **Concatenate strings:**
     - You can use the **append** method to concatenate strings.

```
StringBuilder sb = new StringBuilder();
sb.append("Hello");
sb.append(" ");
sb.append("World");
System.out.println(sb.toString()); // Output: Hello World
```

2. **Insert characters:**
    - You can use the **insert** method to insert characters at a specified index.

```
StringBuilder sb = new StringBuilder("Hello World");
sb.insert(5, ",");
System.out.println(sb.toString()); // Output: Hello, World
```

3. **Delete characters:**
    - You can use the **delete** method to delete characters in a specified range.

```
StringBuilder sb = new StringBuilder("Hello World");
sb.delete(5, 6);
System.out.println(sb.toString()); // Output: Helloorld
```

4. **Replace characters:**
    - You can use the **replace** method to replace characters in a specified range.

```
StringBuilder sb = new StringBuilder("Hello World");
sb.replace(5, 6, ",");
System.out.println(sb.toString()); // Output: Hello, World
```

- It's important to note that the **StringBuilder** class is not thread-safe, meaning it may not work correctly in a multi-threaded environment. If you need a thread-safe alternative, you can use the **StringBuffer** class.

★ Difference between string buffer and string builder in java ?

- the main difference between "StringBuffer" and "StringBuilder" is that "StringBuffer" is thread-safe and "StringBuilder" is not thread-safe. StringBuilder is more performant than StringBuffer.
    - **"StringBuffer"** is a thread-safe class, meaning that it is safe to use in a multi-threaded environment. It creates a separate object for each thread, which prevents conflicts when multiple threads are trying to modify the same string. But it's less performant than StringBuilder
    - **"StringBuilder"** is a non-thread-safe class, meaning that it is not safe to use in a multi-threaded environment. It's more performant than StringBuffer because it doesn't have the overhead of creating a separate object for each thread.

★ StringTokenizer class :-

- used to split a string into tokens (smaller chunks).
- It was deprecated in Java SE 1.5 and replaced by the split() method of the String class.
- Here is an example of how you might use StringTokenizer:

```
StringTokenizer st = new StringTokenizer("Hello World!");
while (st.hasMoreTokens()) {
System.out.println(st.nextToken());
}
This would output:

Hello
World!
```

9) Class :-

- A class is a blueprint for an object.
- It defines the properties and behaviours that an object of that class will have.
- It specifies a set of fields and methods that define the characteristics and behavior of objects of that type.

★ Final Class :-

- a final class is a class that cannot be subclassed or inherited.
- This means that once a class is defined as final, it cannot be extended or modified by any other class.
- A final class is defined by using the keyword "final" before the class name.

★ Abstract Class :-

- A class is said to be abstract class in which its having at least one abstract method.
- You cannot create an object for an abstract class.
- It can contain both abstract and non-abstract methods and can have instance variables.
- Any class that extends an abstract class must implement all the abstract methods defined in the abstract class, unless the subclass itself is also declared as abstract.

★ Inner Class :-

- An inner class is a class that is defined inside another class or interface.
- An inner class has access to the members of the outer class, including its private members.

10) Object :-

- An object is an instance of a class.
- Object is a real-time entity.
- You can create multiple objects of a single class, each with its own unique set of property values.
- An object is a specific occurrence of a class and contains its own data and methods.
- Objects in Java are created using the **"new"** operator, followed by a call to a constructor of the class. Once an object is created, its state can be accessed and modified through its methods.
- For example, consider the following class definition for a Dog:

```
class Dog {
  int age;
  String breed;

  void bark() {
    System.out.println("Woof!");
  }
}
```

- **We can create a Dog object as follows:**

```
Dog myDog = new Dog();
```

- Once the object is created, we can access its state (age and breed) and behavior (bark) as follows:

```
myDog.age = 5;
myDog.breed = "Labrador";
myDog.bark();
```

- Objects in Java are objects in the sense that they are instances of classes, they have a state (represented by instance variables), and they have behavior (represented by instance methods). Java is an object-oriented programming language, and working with objects is a fundamental aspect of programming in Java.

★ How many ways to create object to the class in java ?

- There are **several ways to create** an object of a class in Java:

1) Using the **new keyword.**
2) Using the **Class.forName() method.**
3) Using the **clone() method.**
4) Using the **Class.newInstance() method.**
5) Using **object deserialization.**

1. new Keyword :-

- This is the most common way to create an object in Java.
- It involves using the new keyword, followed by the class name and parentheses.

For example:

```
MyClass obj = new MyClass();
```

2. Class.forName() method :-

- This method returns the Class object associated with the class or interface with the given string name.

For example:

```
Class c = Class.forName("MyClass");
MyClass obj = (MyClass) c.newInstance();
```

3. clone() method :-

- This method creates and returns a copy of the object.
- The class of which the object is an instance should implement the java.lang.Cloneable interface.

For example**:**

```
MyClass obj1 = new MyClass();
MyClass obj2 = (MyClass) obj1.clone();
```

4. Class.newInstance() method :-

- This method creates a new instance of the class represented by this Class object.
- The class should have a public no-argument constructor.
- For example:

```
MyClass obj = (MyClass) MyClass.class.newInstance();
```

5. Using object deserialization :-

- An object can be created by deserializing a serialized object from a file or from a network connection.
- For example:

```
FileInputStream fileIn = new FileInputStream("myclass.ser");
ObjectInputStream in = new ObjectInputStream(fileIn);
MyClass obj = (MyClass) in.readObject();
```

★ How many methods are present in object class ?

- The Object class, which is the parent class of all classes in Java, has several methods that are available to all objects in the Java language.
- The methods present in the Object class are:
  1. **toString():**
     - Returns a string representation of the object.
  2. **equals(Object obj):**
     - Compares this object to the specified object.
  3. **hashCode():**
     - Returns a hash code value for the object.
  4. **clone():**
     - Creates and returns a copy of this object.
  5. **finalize():**
     - Called by the garbage collector before the object is garbage collected.
  6. **getClass():**
     - Returns the runtime class of an object.
  7. **notify(), notifyAll()** and **wait():**
     - These are used for thread synchronization.
  8. **wait(long timeout)** and **wait(long timeout, int nanos):**
     - These are used for thread synchronization.
- Additionally, starting from Java 9, Object class has also added 3 more methods :
  9. default void **forEach(Consumer action)**
  10. default Spliterator **spliterator()**
  11. default Stream **stream()**

---

11) Method :

- Set of instructions is called methods.
- Method is a block of code that performs a specific task.
- Method is nothing but a function inside a class.

★ How many types of methods are present in java ?

- In Java, there are several types of methods that can be defined, including:

1) **Instance methods**:
   - ➢ These methods are associated with an instance of a class and can access and modify the instance variables of the class.
   - ➢ They are called on an object of a class, and can be overridden.
   - ➢ An instance method is a method that is defined inside a class and is associated with an object of the class, rather than the class itself.
   - ➢ An instance method operates on the instance variables of the class and can access and modify the state of the object.
   - ➢ An instance method is defined with the keyword void or any return type, but not static.

- For example, the toString() method in the Object class is an instance method that returns a string representation of an object.

```
class Car {
          private String make;
             private String model;

             public void setMake(String make) {
                this.make = make;
             }
             public void setModel(String model) {
                this.model = model;
             }
             public String toString() {
                return "Make: " + make + ", Model: " + model;
             }
          }
          Car myCar = new Car();
          myCar.setMake("Toyota");
          myCar.setModel("Camry");
          System.out.println(myCar);
```

2) Static methods:

- ➢ static Method is a Method that is shared by all objects of a class. It is declared using the "static" keyword.
- ➢ method that is also associated with the class and not with an instance of the class. This means that you can call a static method directly on the class, without having to create an instance of the class.
- ➢ These methods are associated with the class itself, rather than an instance of the class. They can be called directly on the class, and do not have access to the instance variables of the class.
- ➢ Static method cannot access instance variables.

- For example, the valueOf() method in the Integer class is a static method that returns an Integer object for a given int value.

```
public class MyClass {
  static int staticVariable = 0;

  static void incrementStaticVariable() {
    staticVariable++;
  }
public static void main(String[] args) {
    MyClass.incrementStaticVariable();
    System.out.println(MyClass.staticVariable);
  }
}
```

3) Abstract methods:

- ➢ These methods are declared in an abstract class and do not have a body. They are meant to be overridden by subclasses.
- ➢ As many abstract methods as there are in the super class, all the code should be written in the sub-class.
- ➢ a method that is declared without an implementation is called abstract method.
- ➢ It is declared using the keyword "abstract" and it does not have a body.

- For example, the following is an example of an abstract method

```
abstract class Shape {
                abstract void draw();  //abstract-method
            }
class Circle extends Shape {
                void draw() {
                    System.out.println("drawing circle");
                }
}
```

4) Final methods:

- ➢ These methods cannot be overridden by subclasses, and their behavior is fixed.
- ➢ This means that once a method is defined as final, its implementation cannot be changed by any class that inherits from the class where the final method is defined.
- For example, the getClass() method in the Object class is a final method that returns the runtime class of an object.

```
class Car {
    final void drive(){
        System.out.println("Driving");
    }
}
```

5) Constructors:

- ➢ These methods are used to create and initialize an object of a class. They have the same name as the class, and are called when a new instance of the class is created.
- For example, the following is a constructor of the class Car

```
class Car {
    private String make;
    private String model;
    public Car(String make, String model) {
        this.make = make;
        this.model = model;
    }
}
Car myCar = new Car("Toyota", "Camry");
```

6) Synchronized methods:

- ➢ These methods are used to control access to shared resources and prevent race conditions. They can be applied to both instance and static methods.

```
class Counter {
    private int count;

    public synchronized void increment() {
        count++;
    }
}
```

7) Native methods:

- ➢ These methods are written in a language other than Java, usually C or C++, and are used to access the underlying operating system or hardware.

8) Default methods:

- ➢ These methods are introduced in Java 8, they have a default implementation and can be overridden by subclasses.

9) Static Initializer:

- ➢ These methods are used to initialize static variables, they are executed when the class is loaded and only once.
- This is not an exhaustive list, but it covers some of the main types of methods that can be defined in Java.

★ Can you Override a Private or static method in java ?

- You cannot Override a Private or Static method in java.
- **A. private method :**
  - ➢ is a method that is only accessible within the same class in which it is declared. Because it is not visible to subclasses, they cannot override it.
- **B. static method :**
  - ➢ As it's not associated with any instance of the class, it can't be overridden by any subclass.

★ Lambda Expression :-

- Lambda expression in Java is a new feature introduced in Java 8, which allows you to write more concise and expressive code by providing a way to pass behavior to a method. It is a shorthand way of writing anonymous functions that can be used to implement functional interfaces in Java.
- Lambda expressions are made up of two parts:

1 **The parameters -** they define the input for the expression.

2 **The body -** which defines the output of the expression.

- Lambda expressions are commonly used with functional interfaces, which are interfaces that have a single abstract method. This makes them ideal for implementing the behavior that will be passed to a method, making the code much more concise and readable.

```
//program to demonstrate lambda expressions
interface test
{
   void add(int x,int y);
}
class test1
{
            static void execute(test t,int x,int y)
            {
                        t.add(x, y);
            }
}
/* lambda use chesthunnappudu kindha code rayakkarledhu
class subclass implements test
{
  public void add(int x,int y)
  {
   int r = x+y;
   System.out.println("result is :"+r);
  }

} */

class LambdaEx
{
   public static void main(String arr[])
   {
      test t1 = (int x,int y) -> {int r =x+y; System.out.println("t1 result is :"+r);};
      test1.execute(t1,10,20);

      test t2 = (int x,int y) -> {int r =x+y; System.out.println("t2 addition result is :"+r);};
      test1.execute(t2,30,20);
      }
}
```

12) Constructors :

- A constructor is a special type of method that is used to create an object.
- It has the same name as the class and is called when you create an object of that class using the "new" keyword.
- Constructors have no return type, not even void, and they are called automatically when an object is created.
- In the Constructor We assign or initialize the values for instance variables.
- Constructors **are two types :-**

A. Default Constructor :-

- A default constructor is a constructor that has no parameters.
- If a class doesn't explicitly define any constructors, then the compiler will automatically provide a default constructor for the class.
- The default constructor has no implementation and it does not initialize any fields.

B. Parameterized Constructor :- (User Defined)

- A parameterized constructor is a constructor that takes one or more parameters.
- These parameters can be used to initialize the object's fields when it is created.
- A class can have multiple parameterized constructors with different sets of parameters.

★ Constructor-Overloading :

- In a class having two or more constructors having same name and different parameters.
- When a class has multiple constructors, the appropriate constructor is called based on the number and types of arguments passed to the constructor. This process is called constructor overloading.

★ Why we use constructors in java :

- constructors in Java are used to create objects of a class and to initialize their instance variables. When an object is created using the new operator, the constructor of the class is called automatically to perform any necessary initialization.

↵ **Here are some of the reasons why constructors are used in Java:**

2. **Initialization of instance variables:** Constructors are used to set initial values for the instance variables of an object, which can be different for each object.
3. **Defining multiple objects with different initial values:** Constructors can be overloaded, which means that you can define multiple constructors with different parameters in a class. This allows you to create multiple objects of the same class with different initial values.
4. **Class invariants:** Constructors are used to enforce class invariants, which are conditions that must always be true for objects of a class. By initializing the instance variables in the constructor, you can ensure that the class invariants are satisfied for every object of the class.
5. **Object creation:** Constructors are used to create objects of a class. When you use the new operator to create an object, the constructor is automatically called to perform any necessary initialization.

13) Variables :

- In Java, a variable is a named location in memory that stores a value of a specific data type. Variables have a name, a data type, and a value.
- The name is used to refer to the variable in the program, the data type determines the kind of value that can be stored in the variable, and the value is the actual data stored in the variable.
- Java supports **different types of variables, such as:**

1) Instance Variables :-

- instance variable, also known as a non-static field, is a variable that is defined inside a class but outside of any method or constructor.
- The value of an instance variable can be different for each object of the class.

2) Static Variables:-

- a static variable is a variable that is shared by all objects of a class. A static variable is declared with the **static** keyword and it is associated with the class, rather than a specific instance of the class.

3) Final Variable :-

- a final variable is a variable that is declared with the keyword final.
- Once a variable is defined as final, its value cannot be changed.
- A final variable can be assigned a value at the time of declaration or in a constructor. Once the value is assigned, it cannot be reassigned again.

4) Local variables : -

- These variables are defined inside a method or a block and have no default values. They are created when the method or block is executed and are only accessible within that method or block.

5) Parameter variables : -

- These variables are defined as part of a method signature and are used to pass values into the method.

★ Difference between local variables and class variables ?

- **Local variable**
  - ➢ is a variable that is defined within block of code, and can only be accessed within that block.
- **Class variable**
  - ➢ is a variable that is defined at the class level, and can be accessed by all instances of the class, as well as the class itself. Class variables are typically used to store information that is common to all instances of a class, such as a shared counter or configuration setting.

14) This. (Keyword):-

- This keyword is useful in instance variable hiding.
- In Java, this keyword refers to the current instance of the class. It is used to access the instance variables and methods of the current class.
  - ➢ **To access instance variables**: When a local variable has the same name as an instance variable, this keyword can be used to refer to the instance variable.
  - ➢ **To call other constructors of the same class**: The this() call can be used to invoke another constructor of the same class, allowing for a more flexible approach to object initialization.

- **To pass the current object as an argument:** this can be passed as an argument to a method or constructor.
- **To refer to the current object:** this can be used to refer to the current object in order to return it from a method.

- It is important to note that this is a reference to the current object and it can be used like any other reference.

★ Super keyword :-

- The super keyword is most commonly used to access methods or fields of the parent (Super) class that have been overridden(Extended) by the child(Sub) class. It allows you to access the implementation of the parent class while still using the child class object.
- It can be used in several ways:
  - To access methods or fields of the parent class, when the current class has overridden them.
  - To call the constructor of the parent class, using the super() call in the constructor of the child class.
  - To refer to the parent class instance, when the current class is an inner class.
  - To call superclass constructor.
  - Super. keyword is used in only sub-class.

```
class Parent {
          void printMessage() {
            System.out.println("This is a message from the parent class.");
          }
        }
class Child extends Parent {
          @Override
          void printMessage()      // Call the printMessage method of the parent class
         {
            super.printMessage();
            System.out.println("This is a message from the child class.");
          }
        }
public class Main {
          public static void main(String[] args) {
            Child child = new Child();
            child.printMessage();
          }
        }
```

15) Interfaces :-

- Interface is similar to a class which contains all variables and methods but all variables and methods are by default Abstract.
- In Interface all methods are abstract methods.
- The variables in the interface are called static variables.
- Derive the subclass from the interface i.e., we use implements.
- Derive the interface from the interface i.e., we use extends.
- These methods are declared in an abstract class and do not have a body. They are meant to be overridden by subclasses By using public keyword.
- In Java, there are **two types of interfaces:**

1. Functional Interfaces:

- These interfaces have exactly one abstract method declared in them and can also have default and static methods.
- They are used as the basis for lambda expressions and method references in Java 8. An example of a functional interface in Java is java.util.function.Predicate.

2. Markable(marker) Interface : Normal Interface

- A marker interface is an interface with no method or fields.
- A marker interface can be defined as the interface having no data member functions.
- In simpler terms , an empty interface is called marker interface.
- A markable reference allows a program to mark or unmark an object in addition to **weakly referencing** it.
- The mark is typically a boolean value that can be set or cleared by a program, and it can be used to track the state of the object or to prevent it from being garbage collected.
- **Ex**. Serializable, Cloneable.

Both normal and functional interfaces play a crucial role in Java programming, and are used extensively in various applications and frameworks.

```
interface A
{
		int p = 10;  //static variable

		void show();
		void display();
}
class B implements A
{
		public void show()
		{
				System.out.println("Iam show method in B");
		}
		public void display()
		{
				System.out.println("Iam display method in B");
		}
}
class InterfaceEx
{
		public static void main(String[] arr)
 {
				B ob = new B();
				ob.show();
				ob.display();
		}
}
```

★ Explain generalization and serialization in java ?

- **Generalization : -**
  - In java refers to the process of creating a more general class or interface from a specific one.
  - This is typically done through inheritance or the implementation of interfaces.
- **Serialization : -**
  - In Java refers to the process of converting an object's state to a byte stream, and then recreating the object from that byte stream.
  - This allows the object to be easily saved to a file or sent over a network, and later deserialized to be used again.
  - Objects that can be serialized must implement the "Serializable" interface.
  - The process of converting an object to a byte stream is called "Serializing" an object and the process of recreating an object from a byte stream is called "Deserializing" an object.

16) Packages :-

- package is a collection of related classes and interfaces.
- It is used to organize your classes and interfaces in a logical manner.
- To use a class or interface that is part of a package, you must include an **import** statement at the beginning of your source file.
- In Java, there are **two types of packages**:

1. Built-in packages :

- These are the packages that are already included in the Java Development Kit (JDK) and are used for various purposes such as Input/Output, networking, and security. Examples of built-in packages are java.io, java.net, and java.security.

2. User-defined packages :

- These are the packages that are created by the user and are used to group related classes and interfaces together.
- They are used to organize and structure the code in a meaningful way and also to avoid naming conflicts.
- Examples of user-defined packages are com.example.mypackage and org.mycompany.util.

----

17) Access Specifiers(Modifiers) :-

- access modifiers are keywords that you can use to specify the visibility and accessibility of a class, field, method, or constructor**.**
- **There are four access modifiers in Java**:

1. Public :-

- Is visible and accessible from any other class.

2. Private :-

- private is only visible and accessible within the class in which it is declared.
- It cannot be accessed from any other class.

3. Protected :-

- protected is visible and accessible within the package in which it is declared, as well as from subclasses of the class in which it is declared.

4. Default :-

- no access modifier is visible and accessible within the package in which it is declared, but is not visible or accessible from outside the package.

18) What is Exception and Throwable ?

- **Exception** Is an abnormal condition which stope normal execution of the program.
- **Throwable** Is the super class of all errors and exceptions in Java.
- Exceptions are two types there are **Pre-Defined Exceptions** and **User-Defined Exceptions**

★ Exception Handling :-

- Exception handling is a mechanism in Java that allows you to handle runtime errors, such as division by zero or an array index out of bounds, in a controlled and predictable way.
- Exception Handling Keywords :

a) **Try** : to identify the exceptions and throw to Catch.
b) **Catch** : to handle the exceptions.
c) **Throw** : to manually throw exception object.
d) **Throws** : to throw out the exception handling part.
e) **Finally** : to exception certain statements mandatory.

Two types of exceptions :

1. Checked Exception :-

- checked exception is an exception that is checked by the compiler at compile time.
- These are the exceptions that are defined in the throws clause of a method or constructor.
- List of checked Exceptions are :-
  1) IOException
  2) FileNotFoundException
  3) ClassNotFoundException
  4) NoSuchMethodException......etc.(java.lang-package)

2. Unchecked Exception :-

- unchecked exception is a type of exception that is not checked at compile-time.
- This means that the compiler does not require the programmer to either catch or specify the exception.
- List of checked Exceptions are :-
  1) NullPointerException
  2) IllegalArgumentException
  3) IndexOutOfBoundsException
  4) ArithmeticException.

- Checked Exception and Unchecked Exceptions are called **Pre-Defined Exceptions**

★ How many types of errors in java ?

- In Java, there are several types of errors that can occur during the development and execution of a program:

1. **Syntax errors:**
   - These occur when the Java syntax rules are not followed, such as missing a semicolon, using an incorrect operator, etc.
2. ***Compile-time errors:**
   - These occur when the code cannot be successfully compiled, such as a missing class, a mistyped method name, etc.
3. ***Run-time errors:** {is called Exception}
   - These occur when the program is executed and an unexpected situation occurs, such as a divide-by-zero error, an array out of bounds exception, etc.
4. ***Logical errors:**
   - These occur when the program runs without any syntax or runtime errors, but does not produce the expected results, such as a calculation that is not performed correctly, or incorrect usage of a method.

5. **Resource leaks:**
    - ➢ These occur when resources, such as memory, files, sockets, etc., are not properly released after use, leading to performance degradation or other problems.
6. **Concurrent errors:**
    - ➢ These occur when multiple threads of execution in a program interfere with each other in unexpected ways, leading to race conditions, deadlocks, etc.

```
class ExceptionEx1
{
		public static void main(String arr[])
		{
				try {
		System.out.println(1+2);
		System.out.println(4*5);
		System.out.println(6/0);
		}
		catch (ArithmeticException e)
		{
		System.out.println("please don't divide by zero");
		}
		catch(IndexOutOfBoundsException e)
		{

		}
		finally {
		System.out.println("hello");
		}
		}
}
```

★ How to handle multiple exceptions in java?

- you can handle multiple exceptions in a **try-catch block** using a few different techniques.
- multiple exceptions can be handled in a single catch block using the pipe (|) operator. The pipe operator is used to separate multiple exceptions in the catch block.
- **Here is an example:**

```
try {
  // code that may throw an exception
} catch (ExceptionType1 | ExceptionType2 | ExceptionType3 ex) {
  // code to handle the exception
}
```

- Alternatively, multiple catch blocks can be used to handle different types of exceptions separately.
- **Here is an example:**

```
try {
  // code that may throw an exception
} catch (ExceptionType1 ex1) {
  // code to handle ExceptionType1
} catch (ExceptionType2 ex2) {
  // code to handle ExceptionType2
} catch (ExceptionType3 ex3) {
  // code to handle ExceptionType3
}
```

- **It's important to note** that when using multiple catch blocks, the more specific exception must be caught before the more general one, otherwise, the compiler will raise an error.

★ User-Defined Exception :

- user-defined exceptions are exceptions that are created by the programmer to meet specific needs. A programmer can create a user-defined exception by extending the Exception.

- User-defined exceptions extend the **Exception** class or one of its subclasses and provide additional functionality specific to the programmer's needs.

```
class MyException extends Exception
{
        MyException(){
                System.out.println("This competition for 10 years below children only");
        }
        MyException(String str){
                super(str);
        }
}
class UserException
{
        public static void main(String arr[]) throws Exception
        {
                try {
                int age=11;

                 if(age>10)
                         throw new MyException("This is only for 10 years below");
                }
                catch(Exception e)
                {
                        System.out.println("please enter correct num");
                }
                String name="chaandu";
                int len = name.length();
        try {
                if(len>=7)
                        System.out.println("valid name");
                else
                        throw new MyException("please entered minimum seven characters");
        }
        catch(MyException e) {
                System.out.println("wrong name entered");
        }
                         System.out.println("hello");
        }
}
```

★ Difference between pre-defined and user-defined exception

- The main difference between pre-defined and user-defined exceptions is that pre-defined exceptions are already defined in the Java language, whereas user-defined exceptions are created by the programmer.
- Pre-defined exceptions are typically used for handling errors that occur in the Java runtime environment, while user-defined exceptions are used for handling errors specific to the program being developed.
- Some other differences between pre-defined and user-defined exceptions in Java include:
  - ➢ Pre-defined exceptions are part of the Java API and are accessible to all Java programs, while user-defined exceptions are specific to the program in which they are defined.
  - ➢ Pre-defined exceptions are usually thrown by the Java runtime environment, while user-defined exceptions are thrown explicitly by the programmer.
  - ➢ Pre-defined exceptions are used for handling general errors that can occur in any program, while user-defined exceptions are used for handling specific errors that may occur in a particular program.
- In general, both pre-defined and user-defined exceptions are useful in Java programming and can help make code more robust and resilient to errors. Programmers can use pre-defined exceptions to handle common errors, and they can create their own custom exceptions to handle errors that are specific to their program.

19) Thread :-

- Single flow of execution.
- A thread is a lightweight unit of execution that runs concurrently with other threads within a single process.

★ Multi-Threading :-

- To execute multiple threads concurrently, in the same process, as opposed to concurrently executing multiple processes.
- Refers to the ability of a program to create and manage multiple threads of execution, which are lightweight units of processing that can run concurrently within a single process. Each thread has its own stack, program counter and local variables, but they share the same heap, which means they can access the same shared variables and objects.
- Its support **multitasking.**

1. Single-Tasking :-

- Second task or process must wait for the completion of first task.

2. Multi-Tasking :-

- Concurrent or Parallel execution.
- Second task or process need not to be wait for the first task.
- **Two types in multi-tasking** :-

a. Process-based :-

- Heavy weight processing.
- consumes more memory.
- step by step process execution.

b. Thread-based :-

- Light weight processing.
- Consumes less memory.
- At a time, execution.

★ What is thread life cycle in java or Thread States

- a thread goes through several states during its lifetime, known as the thread life cycle. These states are:
  - ➢ **New:** When an instance of the Thread class is created, but the start() method has not been called on it, the thread is in the new state.
  - ➢ **Runnable:** After the start() method is called on a thread, it becomes runnable. In this state, the thread is waiting for the operating system to allocate a processor for it to run on.
  - ➢ **Running:** Once a processor is allocated, the thread becomes running and begins to execute its task.
  - ➢ **Blocked/Waiting:** A running thread can enter a blocked or waiting state if it needs to wait for a resource to be available or for some other event to occur.
  - ➢ **Terminated/Dead:** Once a thread completes its task, or if it is interrupted by another thread, it enters the terminated state.

★ What is Thread Priority in java ?

- Thread priority is an integer value that determines the order in which threads are scheduled to run by the operating system.
- The thread scheduler uses the priority value to decide which thread should run next when multiple threads are waiting for execution.
- Maximum Priority is 10
- Minimum Priority is 1
- Normal Priority is 5 which is default.

★ What is sleep or suspend a thread in java ?

- A thread can be temporarily suspended or put to sleep using the **sleep()** method of the **Thread** class.
- The **sleep()** method causes the currently executing thread to wait for a specified amount of time before resuming execution.

```
public class SleepExample {
  public static void main(String[] args) {
    System.out.println("Starting thread...");
    try {
      Thread.sleep(3000);              // Wait for 3 seconds
    } catch (InterruptedException e) {
      System.out.println("Thread interrupted.");
    }
    System.out.println("Thread resumed.");
  }
}
```

- It's important to note that the **sleep()** method can be interrupted by another thread using the **interrupt()** method. If the **sleep()** method is interrupted, it throws an **InterruptedException**. The **InterruptedException** should be handled appropriately in the catch block.
- The **sleep()** method should be used with caution, as it affects the performance of the program and can lead to race conditions and synchronization issues if not used correctly. In general, it's better to use other synchronization methods such as wait, notify, and notifyAll to synchronize the behavior of multiple threads.

20) What is Synchronization in java ?

- Synchronization in Java is a mechanism used to ensure that only one thread can access a shared resource at a time.
- When multiple threads access the same resource simultaneously, there is a possibility of data inconsistency and corruption, which can lead to incorrect program behavior.
- Two main mechanisms for synchronization:

1. **The synchronized keyword:** This keyword can be used to declare a method as synchronized. When a synchronized method is called, the thread that calls the method acquires a lock on the object that the method belongs to. Other threads that attempt to call the same method on the same object will be blocked until the first thread releases the lock.
2. **The synchronized block:** This is a block of code surrounded by the synchronized keyword and an object reference. The synchronized block acquires a lock on the specified object, and other threads that attempt to execute the same block of code on the same object will be blocked until the first thread releases the lock.

21) The Collection Framework :- (Java.util package)

- Set of interfaces and classes.
- A collection is used to store a collection of objects and perform operations on them.
- Collection of classes which handling group of Objects is called Collection Framework.

* <T> means

Which Data-type.

Interface Type	Implementation Class
Set<T>	HashSet<T> LinkedHashSet<T>
List<T>	Stack<T> LinkedList<T> ArrayList<T> Vector<T>
Queue<T>	LinkedList<T>
Map<k,v>	HashMap<k,v> Hashtable<k,v>

A. Set<T> :-

- It is an un-ordered collection of objects, in which duplicate values cannot be stored.
- It grows dynamically.

A.1) HashSet<T> :-

- It stores its elements in a hash table, which is a data structure that uses a hash function to map keys to indices in an array.
- it is used to store a collection of unique elements.
- Here are some of the methods provided by the **HashSet**
- **add(E e):** Adds the specified element to the set if it is not already present.
- **clear():** Removes all of the elements from the set.
- **contains(Object o):** Returns true if the set contains the specified element.
- **isEmpty():** Returns true if the set contains no elements.
- **iterator():** Returns an iterator over the elements in the set.
- **remove(Object o):** Removes the specified element from the set if it is present.
- **size():** Returns the number of elements in the set.

These are some of the common methods provided by the HashSet class in Java. It's important to note that the HashSet class does not guarantee the order of its elements, as it uses a hash table to store its elements. If you need to maintain the order of the elements, you can use the LinkedHashSet class instead.

- **Here is an example of how you might use a HashSet in Java:**

```
//program to demonstrate hashset in java
import java.util.*;
class HashsetEx
{
   public static void main(String arr[])
{
      HashSet<String> names = new HashSet<String>();
```

```
        System.out.println("names.isEmpty() :"+names.isEmpty());

            //adding elements/objects
            names.add("chandu");
             names.add("raparthi");

              if(names.isEmpty())
                System.out.println("names hashset still empty");
              else
               System.out.println("size of the hashset names :"+names.size());
              if(names.contains("raparthi"))
              names.remove("raparthi");
                System.out.println("updated size of the hashset names :"+names.size());

                  names.clear();

        System.out.println("names.isEmpty() after clear :"+names.isEmpty());

                names.add("chandu");
                names.add("vani");
                names.add("sai");
                names.add("pawan");
               names.add("ram");

        System.out.println(names);

            /* for(String n : names)  //for each loop
            {
            String newvalue = "welcome  " +n;
            System.out.println(newvalue);
            } */
            Iterator<String> newNames = names.iterator(); //Iterator loop
            while(newNames.hasNext())
            {
            String n = newNames.next();
            String newvalue = "welcome  " +n;
            System.out.println(newvalue);
            }
            }
}
```

A.2) LinkedHashSet<T> :-

- Same as like HashSet and its subset of HashSet.
- The only difference is ordering guarantee.
- The main difference between a HashSet and a LinkedHashSet is that a LinkedHashSet maintains the order in which elements were added to the set.
- This means that when you iterate over the elements in a LinkedHashSet, they will be returned in the order in which they were inserted.

B. List<T> :-

- Same as Set but it allows duplicate elements.

B.1) Stack<T> :-

- Last-In-First-Out (LIFO) data structure that is used to store a collection of objects.
- It provides four main operations:

1. Push() → The push operation adds an element to the top of the stack.

2. Pop() → pop removes and returns the element at the top of the stack.

3. Peek() →it returns top of the element in stack.

4. Search() → to search the element which is in stack.

```
//program to demonstrate stack in java
import java.util.*;
class stackex
{
                        public static void main(String arr[])
                        {
                           Stack<Integer> marks = new Stack<Integer>();

                           //pushing elements in stack
                           marks.push(99);
                           marks.push(98);
                           marks.push(97);
                           marks.push(95);
                           marks.push(96);
                           marks.push(87);

                           if(marks.empty())//it check stack is an empty or not
                                        System.out.println("stack is empty");
                           else
                                        System.out.println("top of the element in stack :"+marks.peek());//return element

                           System.out.println("the 95-element position in stack is:"+marks.search(95));

                           marks.pop();
                           System.out.println("top of the element in stack after pop :"+marks.peek());
                        }
}
```

B.2) LinkedList<T> :-

- We do operations on top elements in stack. But in LinkedList we can do operations in whatever way we like.
- which elements are stored in nodes, and each node contains a reference to the next node in the list.
- Here are the methods
  - **add(E e):** Adds the specified element to the end of the list.
  - **clear():** Removes all of the elements from the list.
  - **contains(Object o):** Returns true if the list contains the specified element.
  - **get(int index):** Returns the element at the specified position in the list.
  - **isEmpty():** Returns true if the list contains no elements.
  - **iterator():** Returns an iterator over the elements in the list.
  - **remove(Object o):** Removes the first occurrence of the specified element from the list, if it is present.
  - **remove(int index):** Removes the element at the specified position in the list.
  - **set(int index, E element):** Replaces the element at the specified position in the list with the specified element.
    - **size():** Returns the number of elements in the list.

```
//program to demonstrate linkedlist in java
import java.util.*;

class linkedlistex
            {
                        public static void main(String arr [])
                        {
                           LinkedList<String> List = new LinkedList<String>();

                           List.add("chandu");
                           List.add("sai");
                           List.add("pavan");
                           List.add("vani");
```

```
                List.add("siva");
                List.add("kittu");
                List.add("beem");

                List.add(3, "raparthi"); //ea index dhaggara add cheyyali ani
            List.addFirst("ramu");
            List.addLast("hai");
            System.out.println(List);

            System.out.println("is List contains chandu ?"+List.contains("chandu"));

            System.out.println("first element is : "+List.getFirst());  //list lo unna first String vasthundhi
            System.out.println("last element is : "+List.getLast());
            System.out.println(" element at index 4 : "+List.get(4));  //manaki ea index lo kavalo aa index lo vasthindhi
            System.out.println("no of elements in the list "+List.size());

            for(String s:List)
              System.out.println("Hello "+s+" welcome to java classes ");

            List.remove(2);//ea index lo name remove cheyyalo indicate chesthundhi
              }
        }
```

B.3) ArrayList<T> :-

- Its dynamically growable.
- Just we can store and retrieve the elements whenever we can operations on the elements, we can use LinkedList.
- To store data and retrieve.

```
//program to demonstrate ArrayList in java
import java.util.*;
            class ArrayListEx
            {
                public static void main(String arr[])
                {
                        ArrayList list = new ArrayList();
                        list.add("chandu");
                        list.add("Sai");
                        list.add(1,"pavan");
                        list.add(2,"siva");
                        list.add(3,"kittu");

                        System.out.println(list);

                        System.out.println("is list contains Sai : "+list.contains("sai"));
                        System.out.println("in the list 3 element is : "+list.get(3));
                        System.out.println("size of the list :"+list.size() );

                        ListIterator<String> li = list.listIterator();

                        while(li.hasNext())
                        {
                                String s=li.next();
                                System.out.println("hello"+s+"welcome");
                        }
                }
            }
```

B.4) Vector<T> :-

- Vector is a collection similar to an ArrayList, but it is synchronized (thread-safe).
- This means that multiple threads can access a Vector simultaneously without causing errors or inconsistencies.
- A Vector has all the methods of a List, such as
- **add(E e):** Adds the specified element to the set if it is not already present.
- **clear():** Removes all of the elements from the set.
- **contains(Object o):** Returns true if the set contains the specified element.
- **isEmpty():** Returns true if the set contains no elements.
- **iterator():** Returns an iterator over the elements in the set.
- **remove(Object o):** Removes the specified element from the set if it is present.
- **size():** Returns the number of elements in the set.

Here is an example of how to use a Vector in Java

```
//program to demonstrate vector in java
import java.util.*;

class VectorEx
{
                public static void main(String arr[])
                {
                    Vector<String> list = new Vector<String>(6,3);

                    System.out.println("initial capacity : "+list.capacity());

                    list.add("a");
                    list.add("b");
                    list.add("c");
                    list.add("d");
                    list.add("e");
                    list.add("f");

                    System.out.println("capasity after adding 6 elements :"+list.capacity());
                list.add("g");
                System.out.println("capasity after adding 7th element:"+list.capacity());

                System.out.println("first element in vector list : "+list.firstElement());
                System.out.println("last element in vector list : "+list.lastElement());

                Enumeration<String> en = list.elements();
                while(en.hasMoreElements())
                    System.out.println("hello"+en.nextElement());
                    }
}
```

C. Queue<T> :-

- It is an ordered collection of elements.
- In which elements are added to the end of the queue and removed from the front.
- A Queue follows the "first-in, first-out" (FIFO) principle, meaning that the element that has been in the queue the longest is the next one to be removed.

C.1) LinkedList<T>

- Same as in LinkedList in List<T> Interface.
- Both are same.
- is a linked list implementation of the Queue interface, which provides O(1) time complexity for additions and removals at the front and end of the queue.

D. Map<k,v> :-

- It stores key-value pair. Key is unique
- an object that maps keys to values.
- A Map cannot contain duplicate keys; each key can map to at most one value.

D.1) HashMap<k,v>:

- A hash table-based implementation that provides fast lookups and is unordered.
- Not synchronized.
- Not in order.
- **clear():** Removes all of the key-value mappings from the map.
- **containsKey(Object key):** Returns true if the map contains a mapping for the specified key.
- **containsValue(Object value):** Returns true if the map contains one or more keys that are mapped to the specified value.
- **entrySet():** Returns a set of the mappings in the map.
- **get(Object key):** Returns the value to which the specified key is mapped, or null if the map contains no mapping for the key.
- **isEmpty():** Returns true if the map contains no key-value mappings.
- **keySet():** Returns a set of the keys in the map.
- **put(K key, V value):** Associates the specified value with the specified key in the map. If the map previously contained a mapping for the key, the old value is replaced.
- **remove(Object key):** Removes the mapping for the specified key from the map, if it is present.
- **size():** Returns the number of key-value mappings in the map.
- **values():** Returns a collection of the values in the map.

```
import java.util.*;
class MapsEx
        {
                public static void main(String arr[])
                {
                        HashMap<String,String> maps = new HashMap<String,String>();

                        System.out.println("is hashmap is empty : "+maps.isEmpty());
                        maps.put("fl", "telugu");
                        maps.put("sl", "english");
                        maps.put("tl", "hindi");

                        if(maps.isEmpty())
                                System.out.println("map still empty");
                        else
                                System.out.println("maps size is : "+maps.size());
                               System.out.println("keyset() : "+maps.keySet());
                              System.out.println("value: "+maps.values());
                              System.out.println("entry set : "+maps.entrySet());
                              System.out.println("value of the key tl is: "+maps.get("tl"));
                }
        }
```

D.2) Hashtable<k,v>:

- Same to same as like HashMap.
- its Synchronized
- An implementation that is synchronized, making it thread-safe, but slower than other implementations. It is now considered outdated and has been largely replaced by ConcurrentHashMap.

★ What is iterator, list iterator, enumeration

- In Java, an iterator, list iterator, and enumeration are all interfaces used to traverse (or loop through) a collection of objects. Each interface provides a different set of methods and functionalities to iterate over the collection.
    - ➢ **Iterator:**
        - ✓ The Iterator interface allows you to traverse a collection of objects in a forward-only direction. It provides three methods: hasNext(), next(), and remove(). The hasNext() method returns true if there are more elements in the collection, the next() method returns the next element in the collection, and the remove() method removes the last element returned by the iterator.
    - ➢ **ListIterator:**
        - ✓ The ListIterator interface extends the Iterator interface to allow bidirectional iteration. In addition to the methods provided by the Iterator interface, the ListIterator provides two additional methods: hasPrevious() and previous(). The hasPrevious() method returns true if there are previous elements in the collection, and the previous() method returns the previous element in the collection.
    - ➢ **Enumeration:**
        - ✓ The Enumeration interface is similar to the Iterator interface, but it is an older interface that has been largely replaced by the Iterator interface. It provides two methods: hasMoreElements() and nextElement(). The hasMoreElements() method returns true if there are more elements in the collection, and the nextElement() method returns the next element in the collection.

It's important to note that the Iterator and ListIterator interfaces are part of the Java Collections Framework, while the Enumeration interface is part of the Java legacy collections. The Collections Framework is a set of interfaces and classes in Java that provide a standardized way to handle collections of objects.

★ What is the difference between collection and collections ?

- The main difference between the Collection and Collections is that Collection is an interface and Collections is a class.
- **Collection :-**
    - ➢ is an interface which provides a contract for collections classes to implement, it defines the common methods that all collection classes should have.
- **Collections :-**
    - ➢ is a utility class that provides common operations that can be performed on a Collection such as sorting, searching, reversing, and shuffling elements. It provides a set of utility methods for various collection classes like **List**, **Set** or **Queue**.
- In summary, Collection interface defines the common methods that all collection classes should have, while Collections is a utility class that provides a set of utility methods for working with collections that implement the Collection interface.

22) What is garbage collector in java ?

- When an object is no longer being used by a program, it becomes eligible for garbage collection.
- Automatically frees up memory by removing objects that are no longer needed by a program
- The garbage collector periodically scans the heap to determine which objects are no longer being used.
- When it finds such objects, it marks them for garbage collection and then reclaims the memory used by them.
- The garbage collector uses a technique known as "mark and sweep" to identify and reclaim objects that are no longer in use.
- The mark phase identifies all objects that are still in use,
- and the sweep phase reclaims the memory used by all objects that are not marked.
- There is also **different collector that are available** that are specialized on different use case, like :

➢ **Serial Collector :** suitable for single-threaded and small memory

➢ **Parallel Collector :** suitable for multi-threaded, medium to large memory

➢ **CMS(Concurrent Mark-Sweep):** suitable for multi-threaded, medium to large memory

➢ **G1 (Garbage-First) :** suitable for multi-threaded, large memory

23) What is public static void main in java ?

- In Java, **public static void main**(String[] args) is the entry point method for a Java program.
- The JVM (Java Virtual Machine) looks for this method when it starts running a Java program and begins executing the code within it.
- **The keyword public** means that the method is accessible from anywhere and can be called by any other class or code.
- **The keyword static** means that the method is a class method, as opposed to an instance method. This means that it can be called on the class itself, rather than on an instance of the class.
- **The keyword void** means that the method does not return any value.
- **The main method** takes a single argument, an array of strings called **args**, which can be used to pass command-line arguments to the program.

24) Java is a case-sensitive language ?

- Yes, Java is a case-sensitive language.
- This means that the language **differentiates between uppercase and lowercase letters**.
- For example, the variable **x** is not the same as the variable **X**.
- This also applies to keywords, class and method names, and so on.
- For example, the keyword **public** is not the same as **Public** or **PUBLIC**.

25) Why pointers not used in java ?

- They are unsafe.
- Increase the complexity of the program.
- Pointers are a way to directly access memory addresses, which can lead to memory leaks and buffer overflow attacks.
- Java uses references instead of pointers, which are safer and more secure.

26) What is JIT in java ?

- JIT **(Just-In-Time)** Compiler is a component of the Java Virtual Machine (JVM) that improves the performance of Java applications by compiling bytecode into native machine code at runtime.

27) Does "finally" always execute in java ?

- **"Finally,"** block in Java always gets executed, regardless of whether an exception is thrown or not.
- The only exception to this rule is when the JVM exits or the thread executing the try/catch block is terminated. In those cases, the finally block will not be executed.

28) What is object casting in java ?

- In Java, "object casting" refers to the process of converting an object from one type to another.
- This can be done by using the type casting operator by using the "instanceof" operator to check if an object is of a certain type before casting it.

29) What is type casting and type conversion in java ?

- **Type casting :-**
  - ➢ refers to the process of changing a variable from one data type to another, such as from an int to a double.
- **Type conversion :-**
  - ➢ refers to the process of changing the type of a value, such as converting a string to an integer.

❖ Both type casting and type conversion can be done implicitly (automatically by the compiler) or explicitly (using a cast operator or a conversion method).

30) What is up casting and down casting in java ?

- In Java, "up casting" and "down casting" refer to the process of converting an object from a more specific type to a more general type, or vice versa.
  - **Up casting :-**
    - ➢ refers to the process of converting an object from a more specific class to a more general class or interface. This is typically done through inheritance or the implementation of interfaces.
  - **Down casting :-**
    - ➢ refers to the process of converting an object from a more general class to a more specific class. This is typically done by using the type casting operator, but it should be done with caution because it can result in a "ClassCastException" if the actual object is not of the type being casted.

31) How we can achieve abstraction in java ?

- In Java, abstraction can be achieved by using abstract classes and interfaces. An abstract class is a class that cannot be instantiated and is typically used as a base class for other classes.
- An interface is a collection of abstract methods that must be implemented by any class that implements the interface. Both abstract classes and interfaces can be used to provide a common set of methods that can be used by different classes, which is a key aspect of abstraction in programming.

32) Why we go for abstraction in java ?

- Abstraction is a crucial concept in programming that allows developers to create more flexible, maintainable, and reusable code by hiding the complexity and focusing on the overall functionality of the system.

33) Difference between abstraction and abstract in java ?

- Abstraction and abstract in Java are related concepts, but they refer to different things.
- **Abstraction** refers to the process of hiding the implementation details of an object or system and showing only the necessary information to the user.
- **Abstract** is a keyword in Java used to define an abstract class or an abstract method. An abstract class is a class that cannot be instantiated, and it is used as a base class for other classes that implement the functionality declared by the abstract class.

34) Difference between final and finally in java ?

- **"final"** is a keyword that can be used in several contexts. It can be used to declare a variable as a constant, or to prevent a class or method from being overridden.
- **"finally"** is a block of code that is used in a try-catch statement. It is used to ensure that a certain piece of code is always executed, regardless of whether an exception is thrown or caught.

35) Difference between Collection and Array in java ?

- the main difference between a collection and an array is that an array has a fixed size and stores elements of the same type, while a collection is dynamic and can store elements of different types. Collections also provide additional functionality and flexibility, such as the ability to iterate through the elements and perform various operations on them.

36) What's the difference b/w Static, non-static and local variables in java ?

- **static variable** is a variable that belongs to a class rather than an instance of the class. It can be accessed by calling the variable name on the class rather than on an instance of the class.
- **Non-static variables**, also known as instance variables, belong to an instance of a class and can only be accessed through an instance of the class.
- **Local variables** are variables that are defined within a method and are only accessible within that method. They are not accessible outside of the method.

37) Wrapper Class :-

- wrapper class is a class that wraps (encloses) a primitive data type and provides it with additional behaviour and methods.
- It converts primitive datatype to object.
- Wrapper classes provide a way to use primitive data types (such as int, char, and boolean) as objects.
- For example, many Java classes, such as ArrayList, can only store objects and not primitive types. So, if you want to store a primitive type in an ArrayList, you must first convert it to the corresponding wrapper class.
- Additionally, Wrapper classes provide various useful methods for conversion, parsing, and manipulation of primitive types.
- **For example**,
    - **Integer.parseInt(string)** method to convert string to int
    - **Integer.toString(int)** method to convert int to string
    - **Integer.toHexString(int)** method to convert int to hexadecimal string.
- Wrapper classes are also used in autoboxing and unboxing feature in Java, which automatically converts between primitive types and their corresponding wrapper classes.

38) what is boxing and un-boxing in java ?

- **“boxing"** refers to the process of converting a primitive data type (e.g. int, float, boolean) to its corresponding object wrapper class (e.g. Integer, Float, Boolean).
- "**Unboxing**" is the opposite process, where an object wrapper class is converted back to its corresponding primitive data type.

39) what is call-by-value and call-by-reference?

- when a primitive data type (such as int, float, etc.) is passed as an argument to a method, a copy of the value of the argument is made and passed to the method. This is called "call by value".
- when an object is passed as an argument to a method, a reference to the object, not the actual object, is passed to the method. This means that the method can modify the object's state, but it cannot modify the reference itself. This is not called "call by reference" because the reference itself is still passed by value, but rather the ability to modify the state of the object the reference refers to.

❖ ava only has "call by value", but for objects, the reference to the object is passed by value, allowing the method to modify the object's state.

40) what is static block ?

- a static block is a block of code that is executed only once when the class is first loaded into memory.
- It is used to initialize static variables and perform other setup tasks that need to be performed only once for the class as a whole, rather than for each individual object of the class. First priority of execution is static blok
- it is first executed then main method was executed.

41) Command line arguments in java ?

- command line arguments are a way to pass information to a program when it is executed from the command line.
- In Java, the main method of a class can be defined to accept an array of strings as its argument, which can be used to access the command line arguments passed to the program.

42) Files in java ?

- In Java, the java.io package provides classes for reading from and writing to files.
- There are several classes in the java.io package for reading from and writing to files, including:

1. **File:**
   - The File class represents a file or directory on the file system and provides methods for working with files, such as creating a new file, renaming a file, or deleting a file.
2. **FileReader:**
   - The FileReader class is used for reading characters from a file. It provides methods for reading individual characters, arrays of characters, or lines of text from a file.
3. **FileWriter:**
   - The FileWriter class is used for writing characters to a file. It provides methods for writing individual characters, arrays of characters, or strings to a file.
4. **BufferedReader:**
   - The BufferedReader class is used for reading text from a character-input stream, such as a file. It provides methods for reading lines of text from the file.
5. **BufferedWriter:**
   - The BufferedWriter class is used for writing text to a character-output stream, such as a file. It provides methods for writing lines of text to the file.

❖ These classes provide a simple and straightforward way to read from and write to files in Java. It is important to remember to close the file after reading from or writing to it to avoid any resource leaks.

43) What is InputStream and OutputStream ?

- **InputStream:**

➢ The InputStream class is the superclass of all input streams in Java. It provides methods for reading bytes from an input source.

- **OutputStream:**

➢ The OutputStream class is the superclass of all output streams in Java. It provides methods for writing bytes to an output destination.

44) getters and setters in java

- Getters and Setters in Java are methods used to get and set the value of an object's instance variables, also known as properties. The purpose of getters and setters is to provide access to an object's private instance variables while maintaining encapsulation.
- A getter method is used to retrieve the value of an instance variable. The method typically has the same name as the instance variable but with a "get" prefix. For

example, if an instance variable is named "firstName", the getter method would be named "getFirstName".

- A setter method is used to set the value of an instance variable. The method typically has the same name as the instance variable but with a "set" prefix. For example, if an instance variable is named "firstName", the setter method would be named "setFirstName".

```
public class Person {
  private String firstName;
  private String lastName;

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }
}
```

45) What is Enum in java ?

- An Enum in Java is a special data type that lets you define a set of named constants, which are also known as enumeration values.
- Enums are used to represent a fixed set of elements, such as days of the week, months in a year, etc.
- It allows you to specify a type-safe, ordered list of values and is typically used to represent a limited number of options, values that never change.
- Enums are also useful for representing complex structures in a more readable form than integers or strings.
- An Enum is a subclass of the Java class java.lang.Enum.

46) What is bug in java ?

- A bug in Java is an error or flaw in the code that causes unexpected behavior, incorrect results, or system crashes.
- Java bugs can be introduced at any stage of the development process, from writing the code to compiling, testing, and deploying the program.
- Common causes of bugs in Java include syntax errors, logic errors, improper usage of Java classes or libraries, and issues with memory management.
- To resolve a bug, a Java programmer needs to identify the root cause of the problem, diagnose the issue, and apply a solution, which can involve fixing the code, updating dependencies, or implementing new features.
- Debugging tools, such as the Java Debugger, can help Java developers find and fix bugs in their code.

47) What is debugging in java ?

- Debugging in Java is the process of identifying and fixing errors (also known as bugs) in a Java program.
- Debugging helps to ensure that the program runs correctly and meets its specified requirements.
- Java provides several tools and techniques to help with debugging, including:

1. The Java Debugger:

➢ A command-line tool that allows developers to trace the execution of a Java program, set breakpoints, inspect variables, and evaluate expressions.

2. Integrated Development Environments (IDEs):

➢ Most Java IDEs, such as Eclipse and IntelliJ IDEA, provide built-in debugging support, allowing developers to set breakpoints, inspect variables, and step through code execution.

3. System.out.println():

➢ This Java statement can be used to print messages to the console, which can help to identify problems in the code and track its execution.

4. Exception handling:

➢ Java provides an exception handling mechanism that can be used to catch and debug exceptions.

- Debugging can be a time-consuming process, but it is essential to ensure the correct functioning of a Java program. Effective debugging requires a combination of technical skills and problem-solving ability, and is an important aspect of software development.

48) What is a class and an object in Java?

- A class is a blueprint or template that defines the properties and behaviours of objects.
- An object is an instance of a class and is created at runtime. An object has a state and behavior and is used to access the properties and methods defined in a class.

49) What is an abstract class and an interface in Java and how are they used?

- An abstract class is a class that cannot be instantiated but can be subclassed. An abstract class can contain abstract methods, which are methods with no implementation.
- An interface is a blueprint for classes. An interface defines a set of methods that a class must implement. An interface cannot be instantiated, but a class can implement multiple interfaces.

50) What is the difference between an abstract class and an interface in Java?

- An abstract class can contain both abstract and non-abstract methods, while an interface can only contain abstract methods.
- A class can extend only one abstract class, but it can implement multiple interfaces.
- An abstract class can contain instance variables, while an interface cannot.

51) What is the difference between method overloading and method overriding in Java?

- Method overloading allows multiple methods with the same name to be defined in a single class with different parameters.
- Method overriding allows a subclass to provide a new implementation for a method that is already

52) Can you explain the concept of OOPs in Java?

- OOP stands for Object-Oriented Programming, and it is a programming paradigm that is based on the concept of objects. Objects are instances of classes, and they contain both data and methods that operate on that data. OOP is designed to provide a more organized and efficient way to write software.

53) Can you explain the basic structure of a Java program?

- A Java program typically has the following structure:

```
public class MyClass {
  public static void main(String[] args) {
    // Java code goes here
  }
}
```

54) What are the differences between Java and other programming languages?

- Java is a popular object-oriented programming language that has several differences compared to other programming languages. Here are some of the key differences:

Platform independence:

- Java is designed to be platform-independent, which means that the code can run on different platforms without modification.
- This is achieved by using the Java Virtual Machine (JVM), which acts as an interpreter that translates the code into a format that can be executed by the host platform.
- Other programming languages like C and C++ are not platform-independent and require the code to be compiled separately for each platform.

Garbage collection:

- Java has an automatic garbage collection mechanism, which manages memory allocation and deallocation.
- This eliminates the need for manual memory management, which is required in languages like C and C++.
- This makes Java programs more robust and less prone to memory leaks.

Object-oriented programming:

- Java is a pure object-oriented programming language, which means that everything in Java is an object.
- Other programming languages like C and C++ are not pure object-oriented languages and allow for procedural programming.

Standard libraries:

- Java has a large standard library that provides a wide range of functionality, including data structures, networking, I/O, and more.
- This makes it easier for developers to build complex applications without having to reinvent the wheel.

Syntax:

- Java has a syntax that is similar to C and C++, which makes it easy for developers who are familiar with these languages to learn Java.
- Other programming languages like Python and Ruby have a different syntax that may take some time to learn.

Memory management:

- Java manages memory allocation and deallocation automatically, which eliminates the need for manual memory management.
- This makes Java programs less prone to memory leaks and other memory-related issues.

Performance:

- Java programs are generally slower than programs written in other programming languages like C and C++.
- This is because of the overhead associated with the JVM and garbage collection. However, the difference in performance is generally not noticeable for most applications.

ADVANCED JAVA

1) what is advanced java

- Advanced Java is an extension of core Java and refers to the set of advanced programming concepts, tools, and technologies that go beyond the fundamental Java programming concepts.
- Advanced Java technologies are used to build complex, scalable, and robust applications, especially in enterprise environments.
- They offer features such as distributed computing, database connectivity, and web-based application development. Advanced Java developers can leverage these technologies to create high-performance and secure web applications, enterprise applications, and mobile applications.
- Here are some key features of advanced Java:
- **Web-based application development**:
  - ➢ Advanced Java technologies such as Servlets and JSP provide a way to build dynamic and interactive web-based applications.
- **Enterprise application development**:
  - ➢ Advanced Java technologies such as EJB, Spring Framework, and Hibernate provide a set of tools and APIs to develop scalable, transactional, and secure enterprise-level applications.
- **Distributed computing**:
  - ➢ Advanced Java technologies provide features such as Remote Method Invocation (RMI) and Java Message Service (JMS) to enable distributed computing and communication between applications.
- **Database connectivity:**
  - ➢ Advanced Java technologies such as JDBC and Hibernate provide a way to access and manipulate databases in a standard and efficient manner.
- **Security:**
  - ➢ Advanced Java technologies offer a range of security features, such as SSL, digital certificates, and encryption, to ensure the security of applications and data.

- Overall, advanced Java technologies are critical for developing enterprise-level, scalable, and secure applications. They require a strong foundation in core Java programming and an understanding of advanced concepts, tools, and technologies.

2) Advanced java concepts

- Advanced Java refers to the set of concepts and technologies that go beyond the fundamental programming concepts of the Java language. Here are some advanced Java concepts that you might encounter:

1. **JDBC (Java Database Connectivity):**
   - ➢ JDBC is a Java API(Application Program Interface) that enables programmers to access and manipulate databases.
   - ➢ It provides a standard way to connect to and work with relational databases.

2. **Servlets**
   - ➢ The servlet is a Java programming language class used to process client requests and generate dynamic web content.
3. **JSP (Java Server Pages):**
   - ➢ This is web technology that allow developers to create dynamic web pages using Java code.
4. **Enterprise JavaBeans (EJB):**
   - ➢ EJB is a server-side technology that enables developers to create scalable, transactional, and secure business applications.
5. **Spring Framework:**
   - ➢ Spring is an open-source framework that provides a set of tools and APIs for building enterprise-level applications using Java. It includes modules for web development, data access, and security.
6. **Hibernate:**
   - ➢ Hibernate is an object-relational mapping (ORM) framework that provides a way to map Java objects to relational databases. It simplifies the task of working with databases and provides a more object-oriented approach to database access.
7. **Design Patterns:**
   - ➢ Design patterns are reusable solutions to common programming problems. Advanced Java developers often use design patterns to create more robust, flexible, and maintainable code.

- These are just a few examples of advanced Java concepts. Mastery of these and other advanced Java concepts can help developers create more sophisticated and scalable applications.

★ JDBC :-

What is JDBC?

- JDBC (Java Database Connectivity) is a standard Java API used to connect and interact with relational databases.
- It provides a set of classes and interfaces that allow Java applications to execute SQL statements, retrieve and manipulate data, and manage database transactions.

Explain the role of Driver in JDBC.

The JDBC Driver provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each driver must provide implementations for the following classes of the java.sql package:Connection, Statement, PreparedStatement, CallableStatement, ResultSet and Driver.

What is the purpose, Class.forName method ?

This method is used to method is used to load the driver that will establish a connection to the database.

What is the advantage of PreparedStatement over Statement ?

PreparedStatements are precompiled and thus, their performance is much better. Also, PreparedStatement objects can be reused with different input values to their queries.

What is the use of CallableStatement ? Name the method, which is used to prepare a CallableStatement.

A CallableStatement is used to execute stored procedures. Stored procedures are stored and offered by a database. Stored procedures may take input values from the user and may return a result. The usage of stored procedures is highly encouraged, because it offers security and modularity.The method that prepares a CallableStatement is the following: CallableStament.
prepareCall();

Q: What are the steps to connect to a database using JDBC?

A: The steps to connect to a database using JDBC are:

1. Load the JDBC driver using Class.forName() method.
2. Create a connection to the database using the DriverManager.getConnection() method.
3. Create a Statement or PreparedStatement object to execute SQL statements.
4. Execute the SQL statements using the execute() or executeQuery() methods.
5. Process the ResultSet returned by the executeQuery() method.
6. Close the ResultSet, Statement, and Connection objects.

Q: What is a PreparedStatement?

A: A PreparedStatement is a subclass of Statement that is used to execute parameterized SQL statements. It allows you to create a SQL statement with placeholders for parameters, which can be set at runtime using the setXXX() methods. This provides improved performance and security compared to using Statement objects.

Q: What is a ResultSet?

A: A ResultSet is a Java object that represents the result of a SQL query. It provides methods to navigate through the result set, retrieve data, and update the data if necessary.

Q: What is the difference between Statement and PreparedStatement?

A: Statement and PreparedStatement are both used to execute SQL statements, but PreparedStatement is preferred over Statement for the following reasons:

- PreparedStatement allows you to execute parameterized SQL statements, which provides improved performance and security.
- PreparedStatement is precompiled, which means it is optimized for faster execution and can be reused with different parameter values.
- PreparedStatement can be used to execute both DML (Data Manipulation Language) and DDL (Data Definition Language) statements.

Q: What is a connection pool?

A: A connection pool is a cache of database connections that are created and maintained by an application server or a connection pool manager. It is used to improve the performance of applications that require frequent database connections by reducing the overhead of creating and destroying database connections.

Q: What is a DataSource?

A: A DataSource is a Java object that provides a standard interface for obtaining a connection to a database. It is used to simplify the process of connecting to a database by providing a centralized location for configuring database connections.

Q: What is a transaction?

A: A transaction is a logical unit of work that consists of one or more database operations, such as inserting, updating, or deleting data. Transactions ensure data consistency and integrity by guaranteeing that all operations are either committed or rolled back as a single atomic unit.

SERVLETS

What is a Servlet ?

The servlet is a Java programming language class used to process client requests and generate dynamic web content. Servlets are mostly used to process or store data submitted by an HTML form, provide dynamic content and manage state information that does not exist in the stateless HTTP protocol.

Explain the architechure of a Servlet.

The core abstraction that must be implemented by all servlets is the javax.servlet.Servlet interface. Each servlet must implement it either directly or indirectly, either by extending javax.servlet.GenericServlet or javax.servlet.http.HTTPServlet. Finally, each servlet is able to serve multiple requests in parallel using multithreading.

What is the difference between an Applet and a Servlet ?

An Applet is a client-side java program that runs within a Web browser on the client machine. On the other hand, a servlet is a server-side component that runs on the web server. An applet can use the user interface classes, while a servlet does not have a user interface. Instead, a servlet waits for client's HTTP requests and generates a response in every request.

What is the difference between GenericServlet and HttpServlet ?

GenericServlet is a generalized and protocol-independent servlet that implements the Servlet and ServletConfig interfaces. Those servlets extending the GenericServlet class shall override the service method. Finally, in order to develop an HTTP servlet for use on the Web that serves requests using the HTTP protocol, your servlet must extend the HttpServlet instead. Check Servlet examples here.

Explain the life cycle of a Servlet.

On every client's request, the Servlet Engine loads the servlets and invokes its init methods, in order for the servlet to be initialized. Then, the Servlet object handles all subsequent requests coming from that client, by invoking the service method for each request separately. Finally, the servlet is removed by calling the server's destroy method.

What is the difference between doGet() and doPost() ?

doGET: The GET method appends the name-value pairs on the request's URL. Thus, there is a limit on the number of characters and subsequently on the number of values that can be used in a client's request. Furthermore, the values of the request are made visible and thus, sensitive information must not be passed in that way.

doPOST: The POST method overcomes the limit imposed by the GET request, by sending the values of the request inside its body. Also, there is no limitations on the number of values to be sent across. Finally, the sensitive information passed through a POST request is not visible to an external client.

What is meant by a Web Application ?

A Web application is a dynamic extension of a Web or application server. There are two types of web applications: presentationoriented and service-oriented. A presentation-oriented Web application generates interactive web pages, which contain various types of markup language and dynamic content in response to requests. On the other hand, a service-oriented web application implements the endpoint of a web service. In general, a Web application can be seen as a collection of servlets installed under a specific subset of the server's URL namespace.

What is a Server Side Include (SSI) ?

Server Side Includes (SSI) is a simple interpreted server-side scripting language, used almost exclusively for the Web, and is embedded with a servlet tag. The most frequent use of SSI is to include the contents of one or more files into a Web page on a Web server. When a Web page is accessed by a browser, the Web server replaces the servlet tag in that Web page with the hyper text generated by the corresponding servlet.

What is Servlet Chaining ?

Servlet Chaining is the method where the output of one servlet is sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. The last servlet in the chain is responsible for sending the response to the client.

How do you find out what client machine is making a request to your servlet ?

The ServletRequest class has functions for finding out the IP address or host name of the client machine. getRemoteAddr() gets the IP address of the client machine and getRemoteHost() gets the host name of the client machine. See example here.

What is the structure of the HTTP response ?

The HTTP response consists of three parts:

- **Status Code:** describes the status of the response. It can be used to check if the request has been successfully completed. In case the request failed, the status code can be used to find out the reason behind the failure. If your servlet does not return a status code, the success status code, HttpServletResponse.SC_OK, is returned by default.
- **HTTP Headers:** they contain more information about the response. For example, the headers may specify the date/time after which the response is considered stale, or the form of encoding used to safely transfer the entity to the user. See how to retrieve headers in Servlet here.
- **Body:** it contains the content of the response. The body may contain HTML code, an image, etc. The body consists of the data bytes transmitted in an HTTP transaction message immediately following the headers.

What is a cookie ? What is the difference between session and cookie ?

A cookie is a bit of information that the Web server sends to the browser. The browser stores the cookies for each Web server in a local file. In a future request, the browser, along with the request, sends all stored cookies for that specific Web server.The differences between session and a cookie are the following:

- The session should work, regardless of the settings on the client browser. The client may have chosen to disable cookies. However, the sessions still work, as the client has no ability to disable them in the server side.
- The session and cookies also differ in the amount of information the can store. The HTTP session is capable of storing any Java object, while a cookie can only store String objects.

Which protocol will be used by browser and servlet to communicate ?

The browser communicates with a servlet by using the HTTP protocol.

What is HTTP Tunneling ?

HTTP Tunneling is a technique by which, communications performed using various network protocols are encapsulated using the HTTP or HTTPS protocols. The HTTP protocol therefore acts as a wrapper for a channel that the network protocol being tunneled uses to communicate. The masking of other protocol requests as HTTP requests is HTTP Tunneling.

What’s the difference between sendRedirect and forward methods ?

The sendRedirect method creates a new request, while the forward method just forwards a request to a new target. The previous request scope objects are not available after a redirect, because it results in a new request. On the other hand, the previous request scope objects are available after forwarding. FInally, in general, the sendRedirect method is considered to be slower compare to the forward method.

What is URL Encoding and URL Decoding ?

The URL encoding procedure is responsible for replacing all the spaces and every other extra special character of a URL, into their corresponding Hex representation. In correspondence, URL decoding is the exact opposite procedure.

http status code

HTTP status codes are three-digit numbers returned by a web server in response to a request made by a client (such as a web browser or an API client). These codes indicate the status of the requested resource or the outcome of the request.

There are five classes of HTTP status codes:

- Informational (1xx): These codes indicate that the request has been received and is being processed, but that the client should continue to wait for a final response.
- Successful (2xx): These codes indicate that the request has been successfully received, understood, and accepted by the server.
- Redirection (3xx): These codes indicate that the client must take some additional action to complete the request, such as following a redirect.
- Client Error (4xx): These codes indicate that the request made by the client was incorrect or incomplete in some way, and the server was unable to process it.
- Server Error (5xx): These codes indicate that there was an error on the server side that prevented the request from being fulfilled.

Some common HTTP status codes that you may encounter include:

- 200 OK: The request was successful.
- 400 Bad Request: The request was invalid or malformed.
- 401 Unauthorized: The client is not authorized to access the requested resource.
- 404 Not Found: The requested resource could not be found on the server.
- 500 Internal Server Error: There was an error on the server side that prevented the request from being fulfilled.

JSP

What is a JSP Page ?

- A Java Server Page (JSP) is a text document that contains two types of text: static data and JSP elements.
- Static data can be expressed in any text-based format, such as HTML or XML.
- JSP is a technology that mixes static content with dynamicallygenerated content.

How are the JSP requests handled ?

- On the arrival of a JSP request, the browser first requests a page with a .jsp extension. Then, the Web server reads the request and using the JSP compiler, the Web server converts the JSP page into a servlet class.
- Notice that the JSP file is compiled only on the first request of the page, or if the JSP file has changed.The generated servlet class is invoked, in order to handle the browser's request.
- Once the execution of the request is over, the servlet sends a response back to the client.

What are the advantages of JSP ?

The advantages of using the JSP technology are shown below:

- JSP pages are dynamically compiled into servlets and thus, the developers can easily make updates to presentation code.
- JSP pages can be pre-compiled.
- JSP pages can be easily combined to static templates, including HTML or XML fragments, with code that generates dynamic content.
- Developers can offer customized JSP tag libraries that page authors access using an XML-like syntax.
- Developers can make logic changes at the component level, without editing the individual pages that use the application's logic.

What are Directives ? What are the different types of Directives available in JSP ?

- Directives are instructions that are processed by the JSP engine, when the page is compiled to a servlet.
- Directives are used to set page-level instructions, insert data from external files, and specify custom tag libraries.
- Directives are defined between < %@ and % >. The different types of directives are shown below:

- **Include directive:** it is used to include a file and merges the content of the file with the current page.
- **Page directive:** it is used to define specific attributes in the JSP page, like error page and buffer.
- **Taglib:** it is used to declare a custom tag library which is used in the page.

What are JSP actions ?

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. JSP actions are executed when a JSP page is requested. They can be dynamically inserted into a file, re-use JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.Some of the available actions are listed below:

- jsp:include - includes a file, when the JSP page is requested.

- jsp:useBean - finds or instantiates a JavaBean.
- jsp:setProperty - sets the property of a JavaBean.
- jsp:getProperty - gets the property of a JavaBean.
- jsp:forward - forwards the requester to a new page.
- jsp:plugin - generates browser-specific code.

What are Scriptlets ?

In Java Server Pages (JSP) technology, a scriptlet is a piece of Java-code embedded in a JSP page. The scriptlet is everything inside the tags. Between these tags, a user can add any valid scriplet.

What are Decalarations ?

Declarations are similar to variable declarations in Java. Declarations are used to declare variables for subsequent use in expressions or scriptlets. To add a declaration, you must use the sequences to enclose your declarations.

What are Expressions ?

A JSP expression is used to insert the value of a scripting language expression, converted into a string, into the data stream returned to the client, by the web server. Expressions are defined between <% =and %> tags.

What is meant by implicit objects and what are they ?

JSP implicit objects are those Java objects that the JSP Container makes available to developers in each page. A developer can call them directly, without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.The following objects are considered implicit in a JSP page:

- application
- page
- request
- response
- session
- exception
- out
- config
- pageContext

Spring Interview Questions

1 What is Spring Framework, and why is it popular?

- Spring Framework is an open-source, lightweight, and modular framework for developing enterprise applications.
- It provides a comprehensive programming and configuration model for building modern enterprise applications.
- The popularity of Spring Framework can be attributed to its simplicity, modularity, extensibility, and rich set of features.

2 What is dependency injection, and how does Spring Framework implement it?

- Dependency injection is a design pattern used to reduce the coupling between components of an application.
- It is a technique where one object supplies the dependencies of another object. Spring Framework implements dependency injection by providing two types of container: BeanFactory and ApplicationContext.
- The container creates objects, manages their lifecycle, and injects dependencies based on the configuration defined in XML files, Java annotations, or Java-based configuration classes.

3 what is inversion control

- IoC suggests that instead of having an object create and manage its dependencies, the dependencies are passed to the object from an external source.
- This allows for more flexibility and modularity in software design, making it easier to modify and test individual components without affecting the entire system.

4 What is AOP, and how does Spring Framework support it?

- Aspect-oriented programming (AOP) is a programming paradigm that helps to modularize cross-cutting concerns in an application, such as logging, security, and transaction management.

5 What is Spring Boot, and how is it different from Spring Framework?

- Spring Boot is a framework that builds on top of the Spring Framework and provides a streamlined way to create stand-alone, production-grade Spring-based applications.
- Spring Boot simplifies the configuration and setup of Spring-based applications by providing defaults for configuration options and auto-configuration for common features.
- It is designed to reduce the time and effort required to get started with Spring, making it ideal for rapid application development and prototyping.

6 What is the difference between a singleton and a prototype bean in Spring Framework?

- In Spring Framework, a singleton bean is an object that is created only once per application context, and the same instance is shared across multiple requests.
- A prototype bean, on the other hand, is an object that is created each time it is requested by the application, resulting in a new instance being created each time.

Hibernate interview questions and answers

1 What is Hibernate?

- Hibernate is an open-source, object-relational mapping (ORM) framework for Java.
- It provides a powerful and efficient way to map Java objects to relational database tables, making it easier to develop and maintain Java-based applications that interact with databases.

2 What is ORM, and how does Hibernate implement it?

- Object-relational mapping (ORM) is a programming technique that maps objects to relational databases.
- Hibernate implements ORM by providing a layer of abstraction between Java objects and relational databases. It allows developers to work with Java objects instead of writing SQL queries directly.
- Hibernate uses configuration files, Java annotations, or Java-based configuration classes to map Java objects to database tables.

3 What is the difference between Hibernate and JDBC?

- Hibernate is an ORM framework that abstracts the database interaction, whereas JDBC is a Java API for connecting to a database and executing SQL queries.
- Hibernate simplifies database interaction by providing a higher-level API and reducing the amount of boilerplate code required to perform database operations.

4 What is lazy loading, and how does Hibernate support it?

- Lazy loading is a technique used to load data only when it is needed. Hibernate supports lazy loading by loading only the required data from the database when it is requested by the application.
- This reduces the amount of data retrieved from the database, improving performance and reducing memory usage.

5 What is the Hibernate SessionFactory, and how does it work?

- The Hibernate SessionFactory is a thread-safe, immutable cache of compiled mappings for a specific database.
- It is responsible for creating Hibernate Session objects that interact with the database.
- The SessionFactory is typically created at the application startup and kept open for the lifetime of the application.

6 What is HQL, and how does it differ from SQL?

- Hibernate Query Language (HQL) is a database-independent object-oriented query language used to retrieve and manipulate data from a database.
- It is similar to SQL, but it operates on the domain objects instead of the database tables.
- HQL supports entity names, properties, associations, and polymorphism, which are not supported by SQL.

Oracle RDBMS interview question and answers

Q: What is Oracle RDBMS?

A: Oracle RDBMS (Relational Database Management System) is a powerful and popular commercial database management system developed by Oracle Corporation. It is used by organizations to manage their large and complex data sets, and it provides a wide range of features for data storage, retrieval, and analysis.

Q: What is a primary key, and how is it different from a unique key?

A: A primary key is a column or set of columns that uniquely identifies each row in a table. It is used to enforce data integrity and ensure that each row has a unique identifier. A unique key, on the other hand, is a column or set of columns that ensures the uniqueness of the data in a table, but it may allow null values.

Q: What is a database trigger?

A: A database trigger is a special type of stored procedure that automatically executes in response to a specific event, such as an insert, update, or delete operation on a table. Triggers can be used to enforce business rules, audit data changes, and perform complex calculations.

Q: What is an index, and how does it work?

A: An index is a database object used to improve the performance of queries. It is a data structure that stores a sorted copy of the values of one or more columns from a table. When a query is executed that references the indexed column(s), the index is used to quickly locate the matching rows, rather than scanning the entire table.

Q: What is the difference between a view and a materialized view?

A: A view is a virtual table that is created by executing a SELECT statement. It is used to simplify the query complexity and present the data in a different format. A materialized view is a physical table that is created by storing the results of a view query. It is used to improve query performance by precomputing and storing the results of a complex query.

Q: What is the difference between a clustered index and a non-clustered index?

A: A clustered index determines the physical order of the data in a table. It is created on the primary key of the table, and it can only be created on one column. A non-clustered index is a separate structure that stores a sorted copy of the indexed columns. It is created to improve query performance and can be created on one or more columns.

tomcat and interview questions

Q: What is Tomcat?

A: Tomcat is an open-source web server and Servlet container that is used to deploy and manage Java web applications.

Q: What is a Servlet?

A: A Servlet is a Java class that is used to process HTTP requests and responses. It runs inside a Servlet container, such as Tomcat, and provides dynamic content to web applications.

Q: What is a JSP?

A: JSP (JavaServer Pages) is a technology used to create dynamic web pages by embedding Java code into HTML pages. It allows developers to create pages that can be customized at runtime and provide a richer user experience.

Q: What is a connector in Tomcat?

A: A connector is a component in Tomcat that is used to handle communication between the server and clients. It defines the protocol used for communication, such as HTTP, HTTPS, or AJP, and manages the connection between the server and client.

Q: What is the difference between a JAR and a WAR file?

A: A JAR (Java Archive) file is used to package Java classes and other resources, such as images and properties files, into a single archive file. A WAR (Web Archive) file is used to package a web application into a single archive file, including JAR files, Servlet classes, JSP files, and other web resources.

Q: What is the Tomcat Manager Web Application?

A: The Tomcat Manager Web Application is a web-based tool used to manage Tomcat web applications. It allows administrators to deploy, undeploy, start, and stop web applications, as well as monitor their performance and status.

Q: What is the difference between Tomcat and Apache?

A: Tomcat is a Servlet container and web server, while Apache is a web server that supports multiple languages and technologies. Apache can be used to serve static content, while Tomcat is used to process dynamic content.

SQL interview question

1 **What is SQL?**

- SQL (Structured Query Language) is a programming language used to manage relational databases.

2 **What is a primary key?**

- A primary key is a column in a table that uniquely identifies each row. It is used to enforce data integrity and ensure that each row in a table can be uniquely identified.

3 **What is a foreign key?**

- A foreign key is a column in a table that refers to the primary key in another table. It is used to establish relationships between tables.

4 **What is a join in SQL?**

- A join is a SQL operation that combines rows from two or more tables based on a related column between them.

5 **What is a subquery?**

- A subquery is a SQL query nested inside another query. It is used to retrieve data that will be used in the main query.

6 **What is a view in SQL?**

- A view is a virtual table that is based on the result set of a SQL query. It is used to simplify complex queries and make them easier to read and manage.

7 **What is normalization in SQL?**

- Normalization is the process of organizing data in a database to reduce redundancy and dependency. It involves breaking up a large table into smaller, more specialized tables and creating relationships between them.

8 **What is a stored procedure?**

- A stored procedure is a precompiled set of SQL statements that are stored in the database and can be called by an application. It is used to simplify complex database operations and improve performance.

9 **What is an index in SQL?**

- An index is a database object that is used to improve the performance of SQL queries. It is a data structure that allows faster retrieval of data based on the values in one or more columns.

10 **What is a transaction in SQL?**

- A transaction is a sequence of SQL statements that are treated as a single unit of work. It is used to ensure data consistency and integrity in a database, and to provide a way to roll back changes if a problem occurs.

JAVA APPLETS

What is an Applet ?

A java applet is program that can be included in a HTML page and be executed in a java enabled client browser. Applets are used for creating dynamic and interactive web applications.

Explain the life cycle of an Applet.

An applet may undergo the following states:

- Init: An applet is initialized each time is loaded.
- Start: Begin the execution of an applet.
- Stop: Stop the execution of an applet.
- Destroy: Perform a final cleanup, before unloading the applet.

What happens when an applet is loaded ?

First of all, an instance of the applet's controlling class is created. Then, the applet initializes itself and finally, it starts running.

What is the difference between an Applet and a Java Application ?

Applets are executed within a java enabled browser, but a Java application is a standalone Java program that can be executed outside of a browser. However, they both require the existence of a Java Virtual Machine (JVM). Furthermore, a Java application requires a main method with a specific signature, in order to start its execution. Java applets don't need such a method to start their execution. Finally, Java applets typically use a restrictive security policy, while Java applications usually use more relaxed security policies.

What are the restrictions imposed on Java applets ?

Mostly due to security reasons, the following restrictions are imposed on Java applets:

- An applet cannot load libraries or define native methods.
- An applet cannot ordinarily read or write files on the execution host.
- An applet cannot read certain system properties.
- An applet cannot make network connections except to the host that it came from.
- An applet cannot start any program on the host that's executing it.

What are untrusted applets ?

Untrusted applets are those Java applets that cannot access or execute local system files. By default, all downloaded applets are considered as untrusted.

What is the difference between applets loaded over the internet and applets loaded via the file system ?

Regarding the case where an applet is loaded over the internet, the applet is loaded by the applet classloader and is subject to the restrictions enforced by the applet security manager. Regarding the case where an applet is loaded from the client's local disk, the applet is loaded by the file system loader. Applets loaded via the file system are allowed to read files, write files and to load libraries on the client. Also, applets loaded via the file system are allowed to execute processes and finally, applets loaded via the file system are not passed through the byte code verifier.

What is the applet class loader, and what does it provide ?

When an applet is loaded over the internet, the applet is loaded by the applet classloader. The class loader enforces the Java name space hierarchy. Also, the class loader guarantees that a unique namespace exists for classes that come from the local file system, and that a unique namespace exists for each network source. When a browser loads an applet over the net, that applet's classes are placed in a private namespace associated with the applet's origin. Then, those classes loaded by the class loader are passed through the verifier.The verifier checks that the class file conforms to the Java language specification . Among other things, the verifier ensures that there are no stack overflows or underflows and that the parameters to all bytecode instructions are correct.

What is the applet security manager, and what does it provide ?

The applet security manager is a mechanism to impose restrictions on Java applets. A browser may only have one security manager. The security manager is established at startup, and it cannot thereafter be replaced, overloaded, overridden, or extended.

SWING

What is the difference between a Choice and a List ?

A Choice is displayed in a compact form that must be pulled down, in order for a user to be able to see the list of all available choices. Only one item may be selected from a Choice. A List may be displayed in such a way that several List items are visible.
A List supports the selection of one or more List items.

What is a layout manager ?

A layout manager is the used to organize the components in a container.

What is the difference between a Scrollbar and a JScrollPane ?

A Scrollbar is a Component, but not a Container. A ScrollPane is a Container. A ScrollPane handles its own events and performs its own scrolling.

Which Swing methods are thread-safe ?

There are only three thread-safe methods: repaint, revalidate, and invalidate.

Name three Component subclasses that support painting.

The Canvas, Frame, Panel, and Applet classes support painting.

What is clipping ?

Clipping is defined as the process of confining paint operations to a limited area or shape.

What is the difference between a MenuItem and a CheckboxMenuItem ?

The CheckboxMenuItem class extends the MenuItem class and supports a menu item that may be either checked or unchecked.

How are the elements of a BorderLayout organized ?

The elements of a BorderLayout are organized at the borders (North, South, East, and West) and the center of a container.

How are the elements of a GridBagLayout organized ?

The elements of a GridBagLayout are organized according to a grid. The elements are of different sizes and may occupy more than one row or column of the grid. Thus, the rows and columns may have different sizes.

What is the difference between a Window and a Frame ?

The Frame class extends the Window class and defines a main application window that can have a menu bar.

What is the relationship between clipping and repainting ?

When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.

What is the relationship between an event-listener interface and an eventadapter class ?

An event-listener interface defines the methods that must be implemented by an event handler for a particular event. An event adapter provides a default implementation of an event-listener interface.

How can a GUI component handle its own events ?

A GUI component can handle its own events, by implementing the corresponding event-listener interface and adding itself as its own event listener.

What advantage do Java's layout managers provide over traditional windowing systems ?

Java uses layout managers to lay out components in a consistent manner, across all windowing platforms. Since layout managers aren't tied to absolute sizing and positioning, they are able to accomodate platform-specific differences among windowing systems.

What is the design pattern that Java uses for all Swing components ?

The design pattern used by Java for all Swing components is the Model View Controller (MVC) pattern.